NASA Contractor Report 198057

On-Line Modal State Monitoring of Slowly Time-Varying Structures

Erik A. Johnson, Lawrence A. Bergman, and Petros G. Voulgaris
Department of Aeronautical and Astronautical Engineering
University of Illinois at Urbana-Champaign
306 Talbot Laboratory, MC-236
104 South Wright Street
Urbana, IL 61801

Prepared for:
NASA Dryden Flight Research Center
Edwards, California
Contract NASA NAG 2-4001

1997

National Aeronautics and
Space Administration

Dryden Flight Research Center
Edwards, California 93523-0273

NOTICE

Use of trade names or names of manufacturers in this document does not constitute an official endorsement of such products or manufacturers, either expressed or implied, by the National Aeronautics and Space Administration.

Available from:

NASA Center for AeroSpace Information
800 Elkridge Landing Road
Linthicum Heights, MD 21090-2934
Price Code: A16

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
Price Code: A16

ACKNOWLEDGMENTS

The authors would like to thank NASA Dryden Flight Research Center for partial support of this work under NASA contract NAG 2-4001.

Special thanks to Mr. Larry Freudinger and Mr. Marty Brenner and their group for bringing this interesting problem to the authors' attention. The collaboration has been greatly appreciated.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

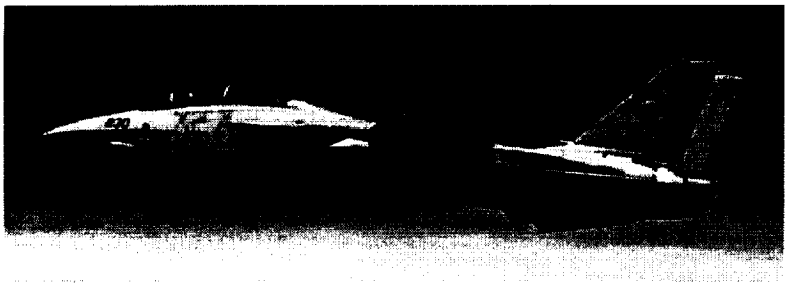
1.0 INTRODUCTION

It may be desirable to monitor the response of structural systems for various purposes. One such purpose may be to monitor stability in order to predict and/or avoid the onset of certain types of pathological behavior; for example, flutter. Another purpose might be to examine information such as frequencies, damping, and response of critical modes. Furthermore, on-line monitoring may be required for detecting structural damage or, conversely, verifying structural integrity. Adaptive on-line control also requires the monitoring of structural response and characteristics.

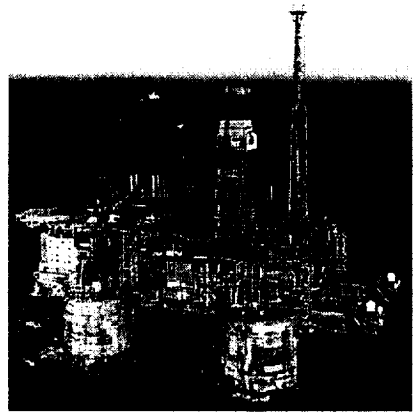
Monitoring structural characteristics is not, however, as straightforward as one might expect, particularly if the goal is real-time, on-line monitoring of slowly time-varying systems with unmodeled dynamics, unknown external forces, rapidly changing control forces, and various noise sources associated with real measurements.

Structures for which this study is applicable include those whose equations of motion couple with those of aerodynamics or hydrodynamics, where the characteristics of the fluid (*e.g.*, pressure, velocity, etc.) may change, causing an effective change in the structural dynamics. Two very different structures that may exhibit such behavior are aircraft and offshore platforms, as shown in Fig. 1.

The efficacy of *modal filters* for monitoring some structural systems was demonstrated by Freudinger (1990, 1991). The applicability of this method to slowly time-varying structures is examined below, followed by a similar examination of several system identification methods, especially H_∞-based identification, and the recommendation of a two-stage adaptive monitoring scheme.



F-14 in flight



The Heidrun tether-leg platform
(350m deep, North Sea near Norway)

Figure 1: Two examples of structures that may vary slowly in time due to coupling with aerodynamic or hydrodynamic phenomena.

2.0 REVIEW OF PROPOSED RESEARCH OBJECTIVES

The dynamics of a linear structural system can be represented in a general manner as

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\dot{\mathbf{x}} + \mathbf{K}\mathbf{x} = \mathbf{f} = \mathbf{f}_k + \mathbf{f}_u \tag{1}$$

where $\mathbf{x}$ denotes the n generalized degrees of freedom and $\mathbf{f}$ is a forcing term that can be decomposed into known and unknown parts, $\mathbf{f}_k$ and $\mathbf{f}_u$, respectively. The known $\mathbf{f}_k$ may include forces that are generated by control actuators, forces due to acceleration, and in general forces that can be directly measured. The unknown $\mathbf{f}_u$ consists of unmodeled and unmeasurable exogenous disturbances (*e.g.*, forces due to turbulence, wind gusts, etc.). Furthermore, it will be assumed that the mass, damping, and stiffness matrices ($\mathbf{M}$, $\mathbf{C}$, and $\mathbf{K}$, respectively) are slowly time varying; that is, they change slowly with respect to the dynamics of the structure.

In most real-world situations, a limited number of sensors are available to collect data. Moreover, sensor noise is invariably present, corrupting the measurements. Therefore, in general the measurement equations can be described by

$$\mathbf{y} = \mathbf{G}\mathbf{z} + \mathbf{v} \tag{2}$$

where $\mathbf{G}$ is an $n_o \times 2n$ matrix, n_o denotes the number of sensors, $\mathbf{z} = [\mathbf{x}^T \ \dot{\mathbf{x}}^T]^T$ the states of the system, and $\mathbf{v}$ the measurement noise. The basic objective of the proposed research was the construction of algorithms which identify and isolate the modes of the system under arbitrary conditions (*e.g.*, an aircraft while in flight) based on a limited number of noisy measurements. Figure 2 shows a block diagram representation of a system using such monitoring algorithms.

There are several additional requirements that must be addressed for a monitoring algorithm to be useful in a real-world problem. The first is that on-line monitoring is typically restricted to less extensive, and often less accurate, testing than a baseline or laboratory test. This may include a limitation on the number, location, and types of sensors. For example, an aircraft can be instrumented thoroughly in a ground test, but in flight one is generally limited to sensors inside the aircraft or embedded within the aircraft structure itself. Additionally, limited data acquisition hardware may further limit the available sensor array.

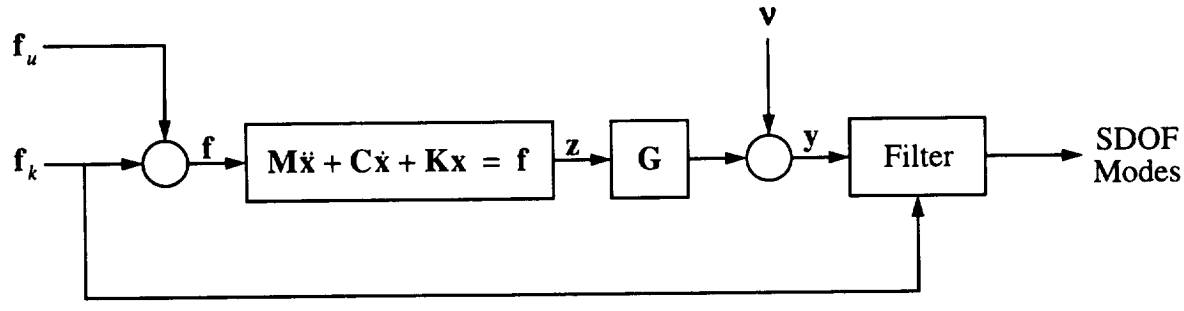


Figure 2: Block diagram representation of the structure and filtered output.

Available computational ability is generally different between laboratory and real-world environments. This certainly must be considered since identification of a complex structure can be computationally intensive. Of course, given a specific level of available computing power, there is generally a trade-off between accuracy of any identified structural parameters and the speed at which those parameters can be updated to track a changing structural system.

There are, then, four basic objectives:

1. Investigate the applicability of *reciprocal modal vectors* to the problem of on-line identification and monitoring of slowly time-varying structures.
2. Explore the various H_∞-based identification algorithms and evaluate their usefulness in on-line monitoring.
3. Analyze methods of modal filtering using other identification algorithms to monitor slowly time-varying structures.
4. Recommend a strategy for on-line monitoring that is robust, accurate, and implementable.

3.0 SPATIAL MODAL FILTERS

Freudinger (1990, 1991) and Shelley (1991) demonstrated the efficacy of the concept of modal filtering via reciprocal modal vectors. They showed that the reciprocal modal vectors perform precisely as modal state observers, that the observer can be constructed from purely experimental data, and that the resulting transformation is relatively insensitive to certain stiffness and damping parameter changes that preserve the structure of the transformation. This appears to hold for data of reasonable quality, even when the input forces are unknown.

However, difficulties arise when perturbations in the identified system parameters, as would occur in an aircraft in flight, affect the transformation. Furthermore, the closed loop behavior of the system may be considerably different from the open loop behavior observed in baseline laboratory test and from which the transformation is derived. Hence, the transformation will, in general, not be preserved.

Thus, while the concept of the modal state observer has been demonstrated, questions remain regarding its use for identification of systems having significant uncertainty and/or time-variance in their physical, control, and input parameters. In order to evaluate the degree of usefulness of the reciprocal modal vector method, its development and theoretical basis will be examined.

3.1 DEVELOPMENT AND USE OF MODAL FILTERS

The modal filter finds its roots in the distributed parameter derivation of *spatial modal filters* by Meirovitch and colleagues in the 1980's (Öz and Meirovitch, 1983, 1984; Meirovitch and Baruh, 1982, 1985; Meirovitch and Ghosh, 1987). The extension to experimentally-determined modal filters was done by Allemang and colleagues in the early 1990's (Zhang, Allemang, and Brown, 1990; Freudinger, 1990, 1991; Shelley, 1991). A summary of these derivations is given below, followed by that of an *ideal* modal filter.

3.1.1 Analytically-Derived Spatial Modal Filters

The basic spatial modal filter derived by Meirovitch (Meirovitch and Baruh, 1982) can be summarized as follows. A self-adjoint (undamped) distributed parameter system can be described by the equation of motion

$$\rho(\mathbf{x})\frac{\partial^2 w(\mathbf{x}, t)}{\partial t^2} + L_K w(\mathbf{x}, t) = f(\mathbf{x}, t), \mathbf{x} \in D \tag{3}$$

where $w(\mathbf{x}, t)$, $\rho(\mathbf{x})$, and $f(\mathbf{x}, t)$ are displacement, density, and force, respectively, at a location $\mathbf{x}$ and time t, and L_K is a linear differential stiffness operator. The system is subject to some boundary conditions

$$L_i w(\mathbf{x}, t) = 0, \mathbf{x} \in \partial D, i = 1, \ldots, p \tag{4}$$

where L_i are boundary operators. The associated eigenvalue problem is given by the differential equations

$$\begin{aligned} L_K W_r(\mathbf{x}) &= \lambda_r \rho(\mathbf{x}) W_r(\mathbf{x}), \qquad \mathbf{x} \in D \\ L_i W_r(\mathbf{x}) &= 0, \quad i = 1, \ldots, p, \quad \mathbf{x} \in \partial D \end{aligned} \qquad r = 1, 2, \ldots . \tag{5}$$

The solution consists of an infinite set of eigenvalues λ_r and associated eigenfunctions $W_r(\mathbf{x})$. Since it is assumed that the stiffness operator L_K is self-adjoint, and further assuming that it is positive definite, then the eigenvalues are all positive, can be ordered such that $\lambda_1 \leq \lambda_2 \leq \ldots$, and are related to natural frequencies by $\lambda_r = \omega_r^2$. Also due to the self-adjointness, the eigenfunctions are orthogonal and can be normalized such that

$$\int_D \rho(\mathbf{x}) W_r(\mathbf{x}) W_s(\mathbf{x}) d\mathbf{x} = \delta_{rs} \tag{6}$$

The system response can then be given by a weighted infinite sum of the eigenfunctions

$$w(\mathbf{x}, t) = \sum_{r=1}^{\infty} \eta_r(t) W_r(\mathbf{x}) \tag{7}$$

and the problem, when substituting (7) into the original equation of motion (3), simplifies to an infinite number of second-order ordinary differential equations in the modal coordinates

$$\ddot{\eta}_r(t) + \omega_r^2 \eta_r(t) = f_r(t), \quad r = 1, 2, \ldots \tag{8}$$

where $f_r(t)$ is a modal force given by

$$f_r(t) = \int_D W_r(\mathbf{x}) f(\mathbf{x}, t) d\mathbf{x} \tag{9}$$

The modal coordinate $\eta_r(t)$ can be found in terms of the displacement $w(\mathbf{x}, t)$ by multiplying (7) by $\rho(\mathbf{x}) W_s(\mathbf{x})$, integrating over D, and using the orthogonality relation (6) to get

$$\eta_r(t) = \int_D \rho(\mathbf{x}) W_r(\mathbf{x}) w(\mathbf{x}, t) d\mathbf{x} \tag{10}$$

Thus, modal displacements can be determined from the physical displacement of the system. Meirovitch recognized, however, that discrete sensors are the most common, so the distributed displacement can be estimated, $\hat{w}(\mathbf{x}, t)$, by fitting eigenfunctions or other interpolation functions to the discrete measurements. This interpolation could also be done using Rayleigh-Ritz or finite element methods.

Meirovitch used this development to derive independent modal space controllers for various problems, including flutter in a very simple bridge model (Meirovitch and Ghosh, 1987) and vibration in a simply-supported Euler-Bernoulli beam (Canfield and Meirovitch, 1994).

3.1.2 Experimentally-Determined Modal Filters

The Enhanced Frequency Response Function (EFRF) technique developed by Allemang (1980) included the basics of the reciprocal modal vector method, though he did not denote it as such at that time. The EFRF technique took weighted sums of frequency response functions (FRFs) to attempt to enhance the amplitude of a mode of interest to better estimate the mode's frequency and damping with single degree of freedom (SDOF) parameter estimation methods.

This method was modified by Zhang, Allemang, and Brown (1990), called the Reciprocal Modal Vector (RMV) method, to determine the *best* weighting coefficients such that the resulting frequency response most closely approaches that of a single degree of freedom system. The extension to multi-input systems was done by Freudinger (1990, 1991), facilitating the use of additional knowledge gained by multi-reference data. Some formalization was done by Shelley (1991), renaming the method the Modified Reciprocal Modal Vector (MRMV) method. He investigated the existence and uniqueness of the reciprocal modal vectors and found that they are not always unique, depending on the number of sensors and other system parameters. Experimental evaluation of the RMV and MRMV methods was performed by Freudinger (1990, 1991) and Shelley (1991).

In its simplest form, the Reciprocal Modal Vector method can be stated as:

> Determine the coefficients of a linear combination of transfer functions from one input to all sensor outputs that results in a minimal difference from a single degree of freedom transfer function with a given pole pair, where "minimal" is determined in a least-square sense.

Shelley does study the issue of spatial distribution of sensors sufficient to accurately determine modal response. It must be noted also that the method requires that the system be *classically* damped. A review of the single-reference development by Shelley (1991) follows.

The transfer function from the q^{th} input to displacement sensor outputs for a real normal mode may be given by a partial fraction expansion

$$\mathbf{H}_q(\omega) = \sum_{r=1}^{n} \left[\frac{\boldsymbol{\phi}_r Q_r \phi_{qr}}{i\omega - \lambda_r} - \frac{\boldsymbol{\phi}_r Q_r \phi_{qr}}{i\omega - \lambda_r^*} \right] \tag{11}$$

where λ_r and λ_r^* are the complex pole pair corresponding to the r^{th} mode, $\boldsymbol{\phi}_r$ is the associated eigenvector, ϕ_{qr} is the modal coefficient of the r^{th} mode at the q^{th} excitation point, and Q_r is a modal scale factor. (If the real normal mode criterion is relaxed, the second term in (11) will have additional * terms, where * represents complex conjugate.) It is then stated that a *reciprocal modal vector* $\boldsymbol{\psi}_s$ exists and can be scaled such that

$$\boldsymbol{\psi}_s^{\mathrm{T}} \boldsymbol{\phi}_r Q_r \phi_{qr} = -i\delta_{rs} = \begin{cases} -i, & r = s \\ 0, & r \neq s \end{cases} \tag{12}$$

This is essentially an orthogonality relation, where the reciprocal modal vector for one mode is orthogonal to all eigenvectors but the one corresponding to the same mode. Premultiplying (11) by $\boldsymbol{\psi}_s^{\mathrm{T}}$ gives

$$\boldsymbol{\psi}_s^{\mathrm{T}} \mathbf{H}_q(\omega) = \sum_{r=1}^{n} \left[\frac{\boldsymbol{\psi}_s^{\mathrm{T}} \boldsymbol{\phi}_r Q_r \phi_{qr}}{i\omega - \lambda_r} - \frac{\boldsymbol{\psi}_s^{\mathrm{T}} \boldsymbol{\phi}_r Q_r \phi_{qr}}{i\omega - \lambda_r^*} \right] \tag{13}$$

Simplifying by using (12) causes all terms but $r = s$ (*i.e.*, all terms but that for the s^{th} mode) to drop out, leaving

$$\boldsymbol{\psi}_s^{\mathrm{T}} \mathbf{H}_q(\omega) = \frac{-i}{i\omega - \lambda_s} + \frac{i}{i\omega - \lambda_s^*} \tag{14}$$

If (14) is evaluated at m distinct frequencies, a matrix equation can be written

$$\left[\begin{array}{c|c|c|c} H_{1q}(\omega_1) & H_{2q}(\omega_1) & \dots & H_{n_o q}(\omega_1) \\ H_{1q}(\omega_2) & H_{2q}(\omega_2) & \dots & H_{n_o q}(\omega_2) \\ \vdots & \vdots & & \vdots \\ H_{1q}(\omega_m) & H_{2q}(\omega_m) & \dots & H_{n_o q}(\omega_m) \end{array}\right] \boldsymbol{\psi}_r = \left\{ \begin{array}{c} \dfrac{-i}{i\omega_1 - \lambda_r} + \dfrac{i}{i\omega_1 - \lambda_r^*} \\ \dfrac{-i}{i\omega_2 - \lambda_r} + \dfrac{i}{i\omega_2 - \lambda_r^*} \\ \vdots \\ \dfrac{-i}{i\omega_m - \lambda_r} + \dfrac{i}{i\omega_m - \lambda_r^*} \end{array} \right\} \tag{15}$$

where each column on the left hand side is a transfer function from one input to an output. This can be written more compactly as

$$\hat{\mathbf{H}}_q^{\mathrm{T}} \boldsymbol{\psi}_r = \boldsymbol{\beta}_r \tag{16}$$

The solution for the reciprocal modal vector $\boldsymbol{\psi}_r$ is not unique if the number of sensors n_o is greater than the number of effective modes of the system. A unique *minimum norm* solution, however, may exist and can be found by using a pseudo-inverse (*e.g.*, a Moore-Penrose generalized inverse), denoted by $(\ )^{\dagger}$

$$\boldsymbol{\psi}_r = [\hat{\mathbf{H}}_q^{\mathrm{T}}]^{\dagger} \boldsymbol{\beta}_r \tag{17}$$

The elements of $\boldsymbol{\psi}_r$ computed in this manner are generally complex numbers, but for real normal modes, this does not make physical sense. Thus the reciprocal modal vector may be restricted to the closest real solution by replacing (17) with

$$\boldsymbol{\Psi}_r = \left[\begin{array}{c} \text{Re } \hat{\mathbf{H}}_q^{\mathrm{T}} \\ \hline \text{Im } \hat{\mathbf{H}}_q^{\mathrm{T}} \end{array}\right]^{\dagger} \left[\begin{array}{c} \text{Re } \boldsymbol{\beta}_r \\ \hline \text{Im } \boldsymbol{\beta}_r \end{array}\right] \tag{18}$$

The extension to multi-input reciprocal modal vectors was done by Freudinger (1990, 1991) and can be computed by replacing (17) with

$$\begin{Bmatrix} \boldsymbol{\Psi}_r \\ -\hat{\phi}_{2r}/\hat{\phi}_{1r} \\ -\hat{\phi}_{3r}/\hat{\phi}_{1r} \\ \vdots \\ -\hat{\phi}_{n_i r}/\hat{\phi}_{1r} \end{Bmatrix} = \begin{bmatrix} \hat{\mathbf{H}}_1^{\mathrm{T}} & 0 & 0 & \cdots & 0 \\ \hat{\mathbf{H}}_2^{\mathrm{T}} & \boldsymbol{\beta}_r & 0 & \cdots & 0 \\ \hat{\mathbf{H}}_3^{\mathrm{T}} & 0 & \boldsymbol{\beta}_r & & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ \hat{\mathbf{H}}_{n_i}^{\mathrm{T}} & 0 & 0 & \cdots & \boldsymbol{\beta}_r \end{bmatrix}^{\dagger} \begin{Bmatrix} \boldsymbol{\beta}_r \\ 0 \\ 0 \\ \vdots \\ 0 \end{Bmatrix} \tag{19}$$

where $\hat{\mathbf{H}}_q^{\mathrm{T}}$ is the FRF matrix from the q^{th} input to all sensors at the m frequencies, and $\hat{\phi}_{sr}$ is the element of the eigenvector $\boldsymbol{\phi}_r$ which corresponds to the s^{th} input degree of freedom. A method similar to that in (18) can be used to restrict the solution of (19) to non-complex solutions.

Removing the restriction of using displacement sensors in the above derivation can be accommodated by redefining the SDOF frequency response function $\beta_r(\omega)$ as

$$\begin{aligned} \beta_r(\omega) &= \frac{-i}{i\omega - \lambda_r} + \frac{i}{i\omega - \lambda_r^*}, && \text{displacement} \\ \beta_r(\omega) &= \frac{\omega}{i\omega - \lambda_r} + \frac{-\omega}{i\omega - \lambda_r^*}, && \text{velocity} \\ \beta_r(\omega) &= \frac{i\omega^2}{i\omega - \lambda_r} + \frac{-i\omega^2}{i\omega - \lambda_r^*}, && \text{acceleration} \end{aligned} \tag{20}$$

This method has been applied to the active modal structural control of a 250-foot span steel truss bridge in Ohio, resulting in a 75% decrease in bridge response (Shelley, Aktan, and Lee, 1994; Shelley, Aktan, Brown, and Allemang, 1994; Shelley, Lee, Aksel, and Aktan; 1995).

3.1.3 The Ideal Modal Filter

Consider the n degree of freedom system given by

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\dot{\mathbf{x}} + \mathbf{K}\mathbf{x} = \mathbf{f} \tag{21}$$

This could be a lumped mass or spatially-discretized distributed parameter system, perhaps discretized via a Rayleigh-Ritz or finite element formulation. The n eigenvalues and eigenvectors of the undamped system can be found via the generalized eigenvalue problem

$$\mathbf{K}\boldsymbol{\phi}_r = \omega_r^2 \mathbf{M}\boldsymbol{\phi}_r \tag{22}$$

Premultiplying (21) by $\mathbf{\Phi}^T$, the transpose of the eigenmatrix

$$\mathbf{\Phi} = [\boldsymbol{\phi}_1 \ldots \boldsymbol{\phi}_n] \tag{23}$$

and substituting $\mathbf{x} = \mathbf{\Phi}\boldsymbol{\eta}$ gives

$$\mathbf{\Phi}^T\mathbf{M}\mathbf{\Phi}\ddot{\boldsymbol{\eta}} + \mathbf{\Phi}^T\mathbf{C}\mathbf{\Phi}\dot{\boldsymbol{\eta}} + \mathbf{\Phi}^T\mathbf{K}\mathbf{\Phi}\boldsymbol{\eta} = \mathbf{\Phi}^T\mathbf{f} \tag{24}$$

Assuming classical damping (which here is equivalent to assuming that the damping $\mathbf{C}$ is a linear combination of the mass $\mathbf{M}$ and stiffness $\mathbf{K}$), the system will decouple into n single degree of freedom equations

$$\ddot{\eta}_r + 2\zeta_r\omega_r\dot{\eta}_r + \omega_r^2\eta_r = \boldsymbol{\phi}_r^T\mathbf{f}, \quad r = 1, \ldots, n \tag{25}$$

and furthermore, $\mathbf{\Phi}^T = \mathbf{\Phi}^{-1}$ such that

$$\boldsymbol{\eta} = \mathbf{\Phi}^T\mathbf{x} \tag{26}$$

which implies that there exists a vector $\mathbf{c}_r$ such that

$$\eta_r = \mathbf{c}_r^T\mathbf{x} \tag{27}$$

Thus, measured displacements $\mathbf{x}$ can be converted directly to the r^{th} modal displacement η_r. Similarly, modal velocities $\dot{\mathbf{x}}$ or accelerations $\ddot{\mathbf{x}}$ can be converted to modal velocities $\dot{\eta}_r$ or accelerations $\ddot{\eta}_r$. If $\mathbf{M}$ and $\mathbf{K}$ are known and all generalized coordinates are measured, then the exact reciprocal modal matrix is $\mathbf{\Phi}^T$.

Given the transfer functions $G_{pq}(\omega)$ from f_q to x_p, then there should exist a $\mathbf{c}_r$ such that

$$\mathbf{G}_q\mathbf{c}_r = \left[\begin{array}{c|c|c|c} G_{1q}(\omega_1) & G_{2q}(\omega_1) & \ldots & G_{n_oq}(\omega_1) \\ G_{1q}(\omega_2) & G_{2q}(\omega_2) & \ldots & G_{n_oq}(\omega_2) \\ \vdots & \vdots & & \vdots \\ G_{1q}(\omega_m) & G_{2q}(\omega_m) & \ldots & G_{n_oq}(\omega_m) \end{array}\right]\mathbf{c}_r = \left\{\begin{array}{c} \dfrac{i}{i\omega_1 - \lambda_r^*} - \dfrac{i}{i\omega_1 - \lambda_r} \\ \dfrac{i}{i\omega_2 - \lambda_r^*} - \dfrac{i}{i\omega_2 - \lambda_r} \\ \vdots \\ \dfrac{i}{i\omega_m - \lambda_r^*} - \dfrac{i}{i\omega_m - \lambda_r} \end{array}\right\} = \boldsymbol{\beta}_r \tag{28}$$

where λ_r and λ_r^* are the complex poles of the r^{th} mode (i.e., $\lambda_r, \lambda_r^* = -\zeta_r\omega_r \pm i\omega_r[1-\zeta_r^2]^{1/2}$) and $\boldsymbol{\beta}_r$ is a constant $(-2\omega_r[1-\zeta_r^2]^{1/2})$ multiplied by the transfer function from f_q to modal displacement η_r evaluated at the m frequencies. If $m = n_o$ (and $\mathbf{G}_q$ is not singular) then $\mathbf{c}_r$ can be computed directly using

$$\mathbf{c}_r = \mathbf{G}_q^{-1}\boldsymbol{\beta}_r \tag{29}$$

Otherwise if $m > n_o$ (*i.e.*, there are more spectral measurement lines than degrees of freedom), a least-squares approximation can be found using a pseudo inverse

$$\mathbf{c}_r = \mathbf{G}_q^{\dagger}\boldsymbol{\beta}_r \tag{30}$$

If the transfer functions $G_{pq}(\omega)$ are instead from f_q to $\dot{x}_p$ or $\ddot{x}_p$, then the right hand side $\boldsymbol{\beta}_r$ should be modified as in (20).

3.2 MODAL FILTER SOFTWARE DEVELOPMENT

A number of MATLAB® programs were written to facilitate the analysis of the Modified Reciprocal Modal Vector method. Primary among these is `mrmv.m`, which is based on that published by Shelley in his Ph.D. disseration (1991), but was reworked for two reasons. Shelley's code was written as a MATLAB® script which is compiled and run line-by-line; it has been replaced by a function which is compiled the first time it is run within a MATLAB® session and subsequently runs significantly faster. Further improvement was made by removing many confusing options and significant user interaction, leading to a yet more efficient and readable code. The necessary options are still available via arguments to `mrmv`.

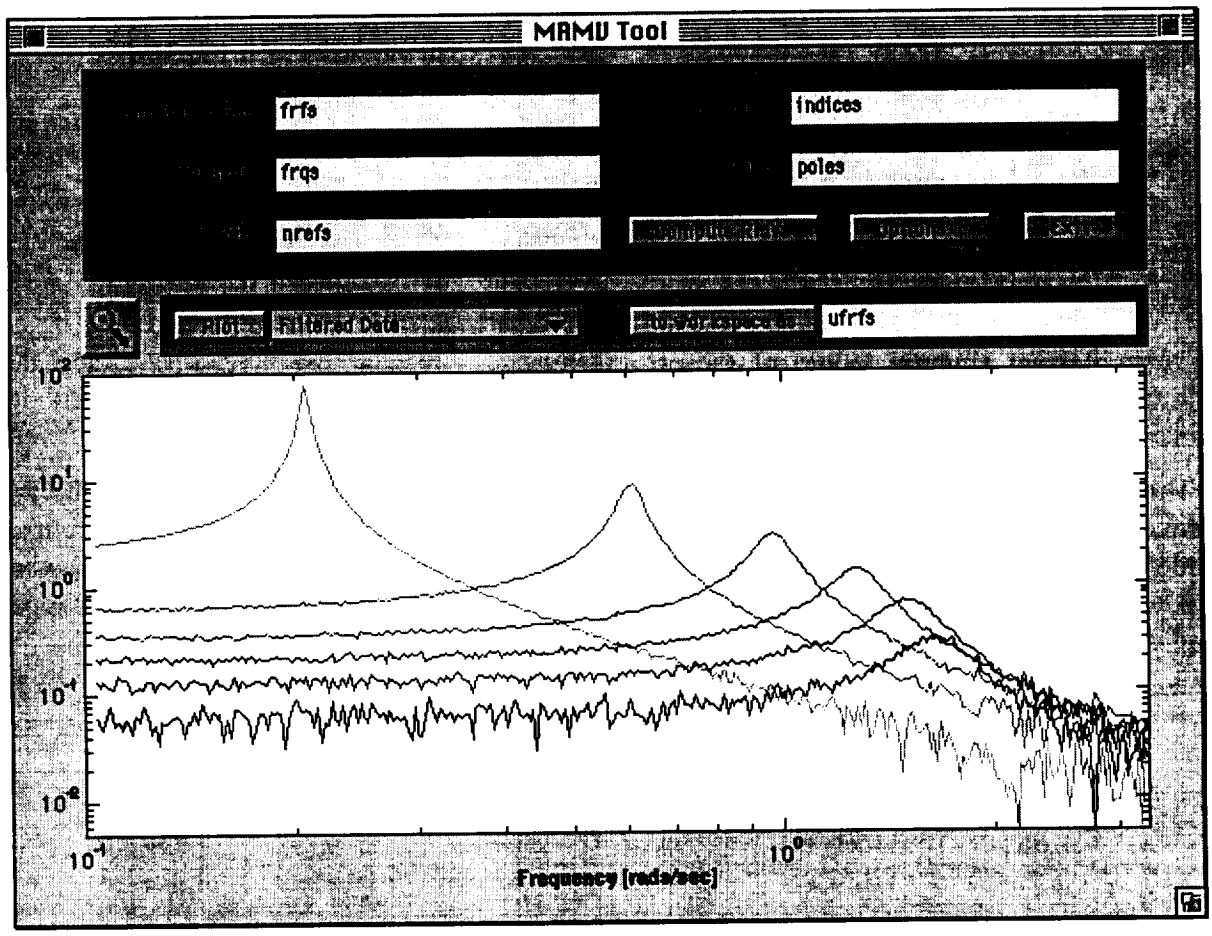


Figure 3: MRMVTool: a graphical user interface for computing the reciprocal modal vectors.

In order to make the use of `mrmv` easier, a graphical user interface (GUI) front-end, called `mrmvtool`, has been coded; a picture of this GUI is shown in Fig. 3.

These programs, as well as most of those used in the following section to evaluate MRMVs, are included in the Appendix.

3.3 Evaluation of Modal Filter Usefulness in On-line Monitoring

To evaluate the usefulness and limitations of the reciprocal modal filter method, the six degree of freedom, six-mode, lumped-mass structure shown in Fig. 4 is used, where sensors measure the absolute displacement of each mass. The effectiveness of the modal filter method is checked by computing the reciprocal modal vectors from a baseline system in which all of the masses, dampers, and springs are identical; *i.e.*,

$$m_i = m, \quad c_i = c, \quad k_i = k, \quad i = 1, \ldots, 6 \tag{31}$$

To simulate an ideal, zero-noise, situation, the exact frequency responses of this system to inputs to each mass are used to compute the matrix of reciprocal modal vectors. The computed reciprocal modal matrix is

$$\Psi_{\text{computed}} = \begin{bmatrix} 0.257782 & 0.456509 & 0.550656 & 0.518654 & 0.367834 & 0.132748 \\ -0.456509 & -0.518654 & -0.132748 & 0.367834 & 0.550656 & 0.257782 \\ 0.550656 & 0.132748 & -0.518654 & -0.257782 & 0.456509 & 0.367834 \\ -0.518654 & 0.367834 & 0.257782 & -0.550656 & 0.132748 & 0.456509 \\ 0.367834 & -0.550656 & 0.456509 & -0.132748 & -0.257782 & 0.518654 \\ -0.132748 & 0.257782 & -0.367834 & 0.456509 & -0.518654 & 0.550656 \end{bmatrix} \tag{32}$$

The norm (largest singular value) of the error $(\Psi_{\text{computed}} - \Psi_{\text{exact}})$ is 1.7821×10^{-13} and the maximum absolute value of the elements of the error is 1.2545×10^{-13}; thus the reciprocal modal vectors are nearly perfect.

This modal filter will then be used to try to decouple the exact transfer functions of the system, with several modifications. Experimentally-measured transfer functions will, of course, include some noise, and the performance of the modal filter would be expected to be somewhat degraded. Thus this noise-free simulation is a best-case scenario and can be considered an upper bound on the modal filter performance.

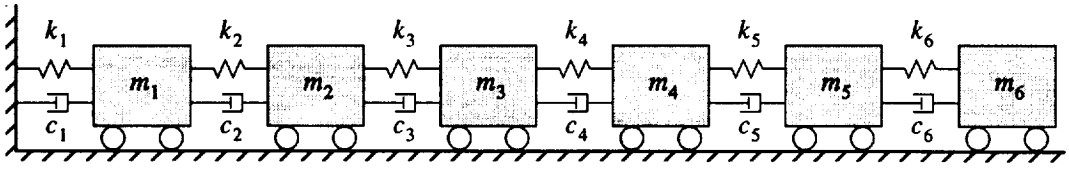


Figure 4: A simple 6 degree of freedom system.

The first system modification to be tested is where all of the masses change by an equal ratio

$$m_i = \mu m \tag{33}$$

The modal filter effectiveness is impervious to such a system change, as can be seen in Fig. 5 which shows the decoupled mode 6 (the other modes all decouple in a like manner). A similar effect is seen when the damping coefficients all change

$$c_i = \chi c \tag{34}$$

such that the reciprocal modal filter also perfectly decouples the modes (Fig. 6).

The same phenomenon does not, however, result when one mass changes. Figure 7 shows the attempt to decouple mode 6 when the 3rd mass has changed sufficiently to cause small shifts in the frequency of mode 6. The corresponding phase plot is shown in Fig. 8, which displays the same rapid degradation of the modal filter effectiveness for even a few percent change in modal frequency. It must be noted that use of acceleration sensors improves the situation somewhat for mode 6 as can be seen in Fig. 9; but other modes display worse results using acceleration sensors rather than displacement sensors.

The common thread throughout is that the reciprocal modal filter works fine when the *modeshape* has not changed. That this should be so is obvious from the derivation above of the *ideal* modal filter from the undamped equation of motion — the ideal modal filter is merely a function of the eigenmatrix of the undamped system. So it may be concluded that the modal filter

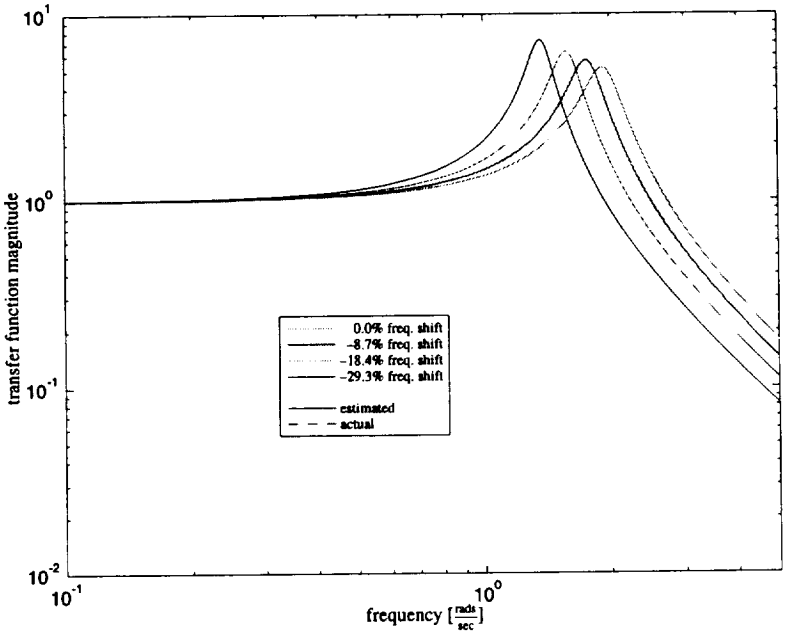


Figure 5: Decoupled mode 6 when all masses change by an equal ratio.

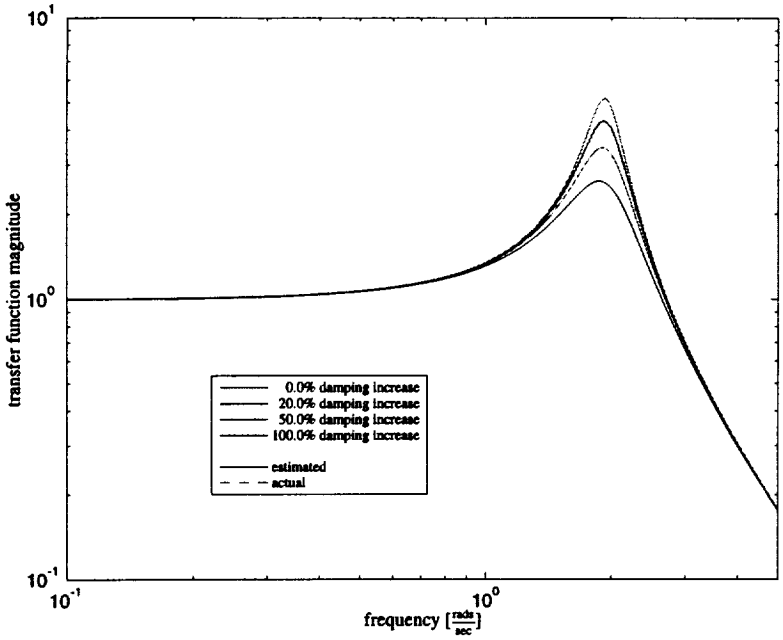


Figure 6: Decoupled mode 6 when all damping coefficients change by an equal ratio.

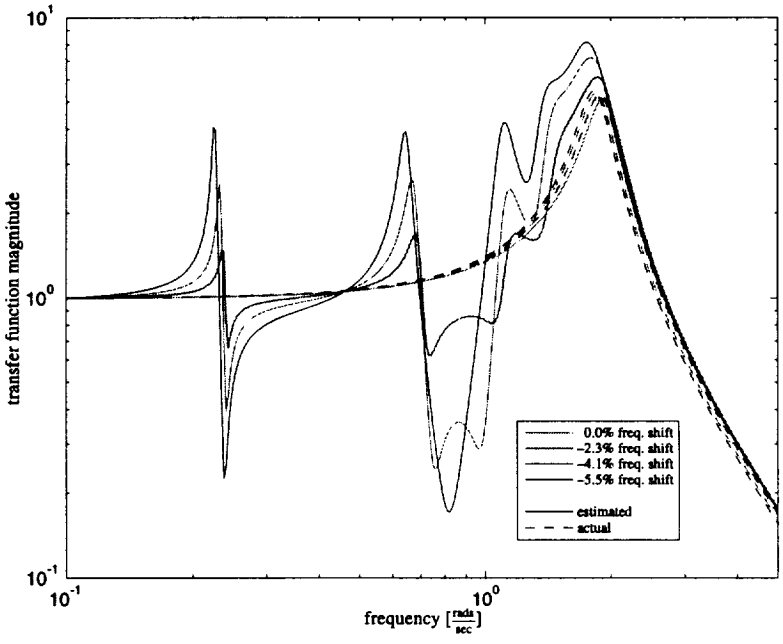


Figure 7: Attempt to decouple mode 6 when one mass changes — transfer function magnitude.

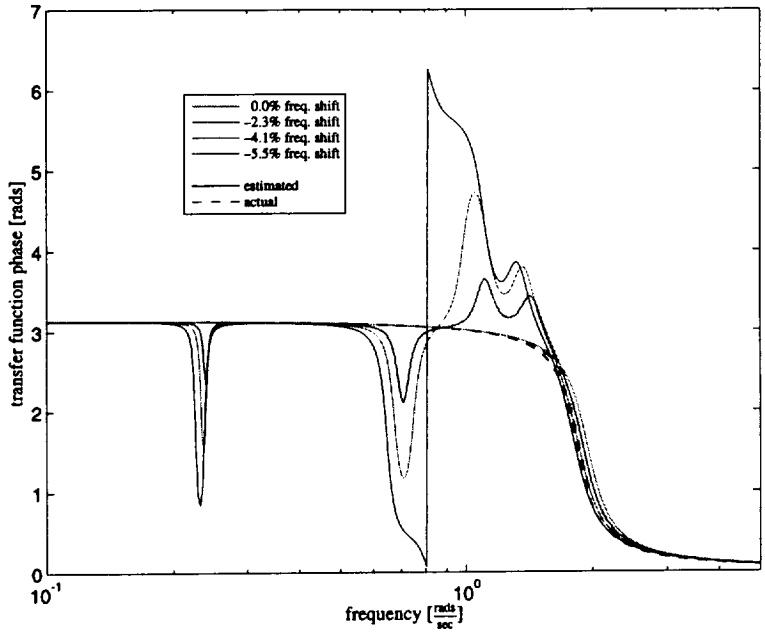


Figure 8: Attempt to decouple mode 6 when one mass changes — transfer function phase.

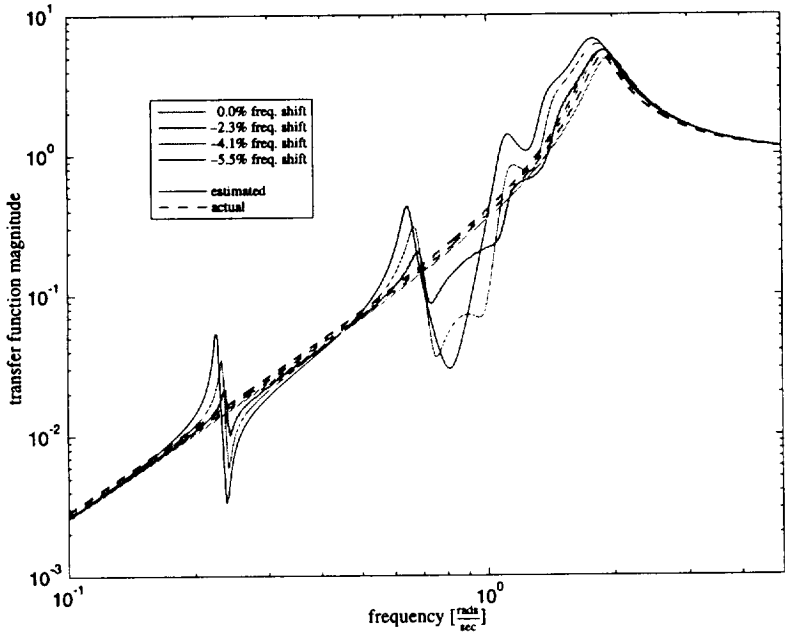


Figure 9: Attempt to decouple mode 6 when one mass changes — acceleration sensors.

is useful for systems whose characteristics are unchanging, or at least whose modeshapes do not change. This criterion may be met by some structural systems, but it lacks the generality desired in this study.

There are other practical limitations to the use of the reciprocal modal vector method for on-line monitoring of slowly time-varying systems. The reciprocal modal vector requires on-line the same array of sensors — the same number, location, and types of sensors — used to compute the reciprocal modal vector. Due to the fact that on-line monitoring is often restricted to a reduced or different array of sensors, this limits the practical usefulness of the reciprocal modal filter method.

Another concern is the issue of uniqueness. The reciprocal modal vectors are not necessarily unique. Scenarios can be constructed in which computed reciprocal modal vectors are significantly shifted from the exact solution. In such situations, using filtered response may produce inaccurate results and non-conservatively estimate damage to the system or the onset of behavior like flutter.

There is also concern about the accuracy of the poles used to compute the reciprocal modal vectors. If these pole estimates are poor, the modal filter will also function poorly. A more robust method is required.

A method to accurately update the reciprocal modal vectors to reflect changes in system modeshapes would be ideal, but tracking eigenvector changes is difficult in general, and particularly so when less than full state information is available (Beck, 1996; Beck and Vanik, 1996).

To summarize, the modal filter works well to monitor systems whose modeshapes are unchanging and that may be fully instrumented. However, to use the reciprocal modal filter to identify the modal poles of a time-varying system is fraught with difficulties in the general case where changes in the system will cause changes in the shapes of the modes, thus invalidating any response filtered by reciprocal modal vectors.

4.0 STRUCTURAL SYSTEM IDENTIFICATION METHODS

There are several ways to classify the various system identification methods that are applicable to structural systems. One division is parametric vs. non-parametric identification; the former parameterizes the system, using a given model structure, with a finite number of unknown parameters, whereas the latter typically models the system as a set of functions, generally the transfer functions themselves, and estimates these functions usually by correlation or spectral analysis. The focus here is on parametric methods since modal characteristics are of primary interest, and because non-parametric methods typically require input-output data, which may not always be available.

Various frequency domain least-squares methods exist to match a parametric model of a transfer function to measured transfer function data. Such methods date back to the work of Levy (1959) who parameterized a continuous-time transfer function by the coefficients of numerator and denominator polynomials, whereas others have used Chebyshev polynomials (*e.g.*, Adcock, 1987). Improvements on these methods have had some limited success; for example, Sanathanan and Koerner proposed an iterative method (1963) that often arrives at a better solution, but is not guaranteed to converge.

Covariance and singular value decomposition methods are another rough class of identification methods. This class includes the Eigensystem Realization Algorithm (ERA), developed by Juang and Pappa (1985), which uses a generalized Hankel matrix of the system Markov parameters to identify system parameters. This algorithm was later extended to handle auto- and cross-correlation data directly (ERA/DC) (Juang, 1994). Another method in this class is q-Markov COVER (Liu and Skelton, 1993), which under certain conditions produces results identical to ERA/DC (Peterson, 1993). The Frequency domain Observability Range Space Extraction (FORSE) method (Jacques, 1994) uses frequency domain data directly, and in the limit as the number of data points tends to infinity gives the same results as q-Markov COVER. The subspace methods, such as N4SID (Numerical algorithms for Subspace State-Space System IDentification) (Van Overschee and De Moor, 1994, 1995; Viberg, 1995), also fall into this classification.

The requirements of H_∞ robust control have motivated recent work in H_∞-based identification. Such control algorithms typically require knowledge of the bounds on the uncertainty in the plant model, but most identification methods cannot provide this information to the H_∞ control designer.

Undoubtedly, the best-known class of identification methods comprises the various time-domain least-squares methods. These are distinguished from other methods by their ease of use and their implementation within software packages such as MATLAB®.

In the sections below, H_∞-based identification will be examined in detail, followed by a cursory examination of the ERA method, the basics of time-domain least squares methods, and finally a brief comparison of a number of methods in light of the problem of on-line monitoring of slowly time-varying structural systems.

4.1 H_∞-BASED SYSTEM IDENTIFICATION

Identification algorithms based on H_∞ methods typically provide the bounds on plant uncertainty required for H_∞ robust control design. Most of these methods are rather similar; a typical derivation and two examples will be given below.

One dissimilar algorithm that deserves mention, is the Noise Perturbed Full State Info (NPSFI) algorithm (Didinsky, Pan, and Basar, 1995), which uses the *cost-to-come* method, developed to solve nonlinear H_∞ optimal control and filtering problems, to identify uncertain plants that are linear in the unknown parameters but nonlinear otherwise. This algorithm, however, requires at least full state information (and in some cases full state derivative information), which effectively disqualifies its usefulness here since full state information is not generally available in on-line monitoring.

4.1.1 Summary of Previous Work

Many developments in the area of robust system identification have occurred over the past decade, highlighted by a Special Issue of *IEEE Transactions on Automatic Control* on System Identification for Robust Control Design (Kosut, Goodwin, and Polis, 1992). The general goal of this work has been to identify a system in the presence of noise and unknown plant dynamics using time- or frequency-response data of the system. Obviously, one desires that the error between the actual system and the identified system goes to zero in some sense. The recent work can generally be divided into two classifications: systems with (*i*) stochastic noise (*e.g.*, white noise or filtered white noise), and (*ii*) deterministic noise with a bounded infinity-norm and unmodeled plant dynamics. Examples of the former approach can be found in Ljung (1987, 1995). Several recent papers (*e.g.*, Bai and Andersland, 1994; Partington and Mäkilä, 1995a), including some in a special issue of *Automatica* on "Trends in System Identification" (Söderström and Åström, 1995), have analyzed stochastic least-squares identification with worst-case identification techniques and have shown that they are by no means mutually exclusive.

One of the earliest references to the bounded deterministic approach is by Zames (1979), who used the theory of metric complexity to study issues related to the complexity of identification. The formulation by Helmicki, Jacobson, and Nett (1989, 1990a, 1990b, 1991c, 1991a, 1991b, 1992), however, provided the concept of robustly convergent identification that is the real foundation for most of the current work in H_∞-based system identification. Essentially, they developed a theory for the robust identification of a system in discrete-time based upon a finite number of frequency response measurements (that may be corrupted by noise); the resulting system approximation converges in the H_∞ sense to the real system as the noise and the number of measurements tend to zero and infinity, respectively.

The algorithms of Helmicki, *et al.* fall into the categories of linear and nonlinear. The linear algorithms are less complex (and therefore less computationally intensive), but require tuning to certain *a priori* information about the unknown system (the convergence properties may fail if the *a priori* knowledge happens to be wrong). Untuned linear algorithms have been developed (*e.g.*, Gu and Khargonekar, 1992a, 1992b), but they are not robustly convergent — in fact, Partington (1991, 1992) showed that no robustly convergent, untuned, linear algorithms exist. Linear H_∞ identification methods can be made robustly convergent in the presence of noise by requiring *a priori* information; for example, Bai and Raman (1994) do so by incorporating a projection operator based on an assumed *a priori* knowledge of an exponential decay bound on the magnitude of the system pulse response (*i.e.*, $|h(k)| \leq M\rho^k$ for $k \geq 0$). The utility of *a priori* probabilistic information has also been studied (Jacobson and Tadmor, 1993).

Gu and Khargonekar (1992a, 1992b), and Partington (1992), building upon earlier work, have developed some rapidly convergent nonlinear algorithms. Many of these algorithms that were originally developed for discrete-time systems have been extended to continuous-time (Akçay,

Gu, and Khargonekar, 1993; Chen, Gu, and Nett, 1994; Helmicki, Jacobson, and Nett, 1990b, 1992; Mäkilä, 1991a).

The relationship of these algorithms to the finite-dimensional approximation of infinite-dimensional systems was investigated (Gu, Khargonekar, and Lee, 1989), as has adaptive system identification based upon frequency response for various systems (*e.g.*, Parker and Bitmead, 1987a, 1987b).

Identification in l_1, while not examined in this study, has been done, notably in several papers by Mäkilä (1991b, 1992) and Partington and Mäkilä (1995b). This may be worth additional study at some point, but it is expected that it will demonstrate many of the same advantages and disadvantages in the context of on-line monitoring of (slowly) time-varying systems as the H_∞-based identification examined herein.

4.1.2 The H_∞-based Identification Algorithms

Gu and Khargonekar summarized the H_∞-based system identification algorithms in two papers (1992a, 1992b). For simplicity, assume that the unknown system to be identified is single-input single-output (SISO). Further, assume that the unknown system is stable, linear, shift-invariant, and discrete-time, with transfer function H. The necessary information is N points of time-domain data, $\hat{\mathbf{h}}(kT)$, $k = 0, \ldots, N-1$ (where T is the sampling time; note that $\hat{\mathbf{h}}(kT)$ is shortened to $\hat{\mathbf{h}}(k)$ for convenience). The time domain data may originate in an inverse discrete Fourier transform of an N-point set of (noisy) experimental frequency response data in the form of

$$\hat{\mathbf{H}}(e^{2i\pi\frac{j}{N}}) = \mathbf{H}(e^{2i\pi\frac{j}{N}}) + \boldsymbol{\eta}(e^{2i\pi\frac{j}{N}}),\ j = 0, \ldots, N-1 \tag{35}$$

where the noise vector is bounded, $\|\boldsymbol{\eta}\|_\infty \le \varepsilon$, and where $i = \sqrt{-1}$. The inverse discrete Fourier transform may be defined by

$$\hat{\mathbf{h}}(k) = \frac{1}{N}\sum_{j=0}^{N-1} \hat{\mathbf{H}}(e^{2i\pi\frac{j}{N}}) e^{-2i\pi(N-k)\frac{j}{N}}. \tag{36}$$

The time sequence $\hat{\mathbf{h}}(k)$ is assumed to be periodic both forward and backward in time, such that $\hat{\mathbf{h}}(k) = \hat{\mathbf{h}}(k \pm mN)$ for any integer m. (Alternate formulations can accommodate a frequency-dependent noise bound by using a frequency-dependent weighting function (Helmicki, Jacobson, and Nett, 1991b) and non-uniform spacing of the frequencies by using an interpolation method (*e.g.*, Akçay, Gu, and Khargonekar, 1992; Partington, 1993).)

The problem then is to find an identified model $\hat{\mathbf{F}}$ of the system $\mathbf{H}$ such that $\|\mathbf{H} - \hat{\mathbf{F}}\|_\infty$ is minimized and such that

$$\lim_{\substack{\varepsilon \to 0 \\ N \to \infty}} \|\mathbf{H} - \hat{\mathbf{F}}\|_\infty = 0 \tag{37}$$

The algorithms by Helmicki, *et al.*, Gu, Khargonekar, Mäkilä, and Partington (see references above) can be divided into two categories: linear and nonlinear. The linear algorithms can be

divided into those that are tuned to *a priori* system information and untuned. The nonlinear and the tuned linear algorithms have been shown to be robustly convergent, but the untuned linear algorithm has been shown to not be robustly convergent (Partington, 1991, 1992), though its divergence is generally so slow that it is often useful. The untuned linear and the two-stage nonlinear algorithms will be summarized below.

Untuned Linear Algorithm

The untuned linear algorithm is quite simple. It involves taking a weighted, one-sided z-transform of the time sequence data

$$\hat{\mathbf{F}}^{n}_{\text{linear}}(z) = \sum_{k=0}^{n} w(k, n)\hat{\mathbf{h}}(k)z^{-k} \tag{38}$$

where $w(k, n)$ is a weighting, or windowing, function. Some of the weighting functions that have been investigated (Gu and Khargonekar, 1992a) are spline, cosine, triangular, rectangular, trapezoidal, and Hamming; these are described by the functions below (where $0 \le m \le n \le M < N$) and are shown graphically in Figure 10.

$$w_{\text{spline}}(k, n) = \begin{cases} 1, & k = 0 \\ \left(\frac{M}{k\pi}\sin\frac{k\pi}{M}\right)^2, & 1 \le |k| \le n \\ 0, & |k| > n \end{cases} \qquad w_{\text{cosine}}(k, n) = \begin{cases} \cos\frac{k\pi}{2n+1}, & |k| \le n \\ 0, & |k| > n \end{cases} \tag{39}$$

$$w_{\text{triangular}}(k, n) = \begin{cases} 1 - \frac{|k|}{n}, & |k| \le n \\ 0, & |k| > n \end{cases} \qquad w_{\text{Hamming}}(k, n) = \begin{cases} 0.54 + 0.46\cos\frac{k\pi}{n}, & |k| \le n \\ 0, & |k| > n \end{cases} \tag{40}$$

$$w_{\text{trapezoidal}}(k, n) = \begin{cases} 1, & 0 \le k \le 2m \\ \frac{n - |k - m|}{n - m}, & m - n \le k \le 0, \text{ or } 2m \le k \le n + m \\ 0, & |k - m| \ge n \end{cases} \qquad w_{\text{boxcar}}(k, n) = \begin{cases} 1, & |k| \le n \\ 0, & |k| > n \end{cases} \tag{41}$$

The error for the untuned linear algorithm has been shown to be divergent as n tends toward infinity. The worst-case identification error (Gu and Khargonekar, 1992b) for a SISO system is bounded below

$$\inf_{\text{admissible } \hat{F}} \left\| H - \hat{F} \right\|_{\infty} \ge \left[\frac{1}{\pi}\log n + A\right]\varepsilon - \alpha_n \tag{42}$$

where A is a finite constant and $\{\alpha_n\}$ is a sequence that tends to zero as n tends to infinity. However, $\log n$ is unbounded for $n \to \infty$; thus, the worst-case error is unbounded. If, however,

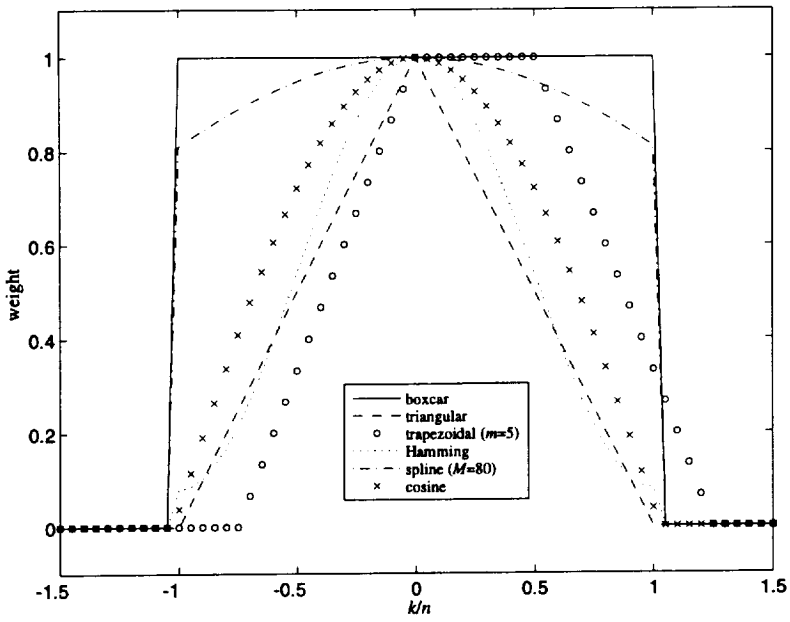


Figure 10: Windowing functions.

the noise bound ε and the required order n are small enough, in practice the results of the linear algorithm are often quite close to that of the nonlinear algorithm (Gu and Khargonekar, 1992b).

Two-Stage Nonlinear Algorithm

The nonlinear algorithm has two steps. The first is much like the untuned linear algorithm above, taking a weighted, but now two-sided, z-transform of the time sequence data. Let $\hat{\mathbf{h}}_w(k)$ be a weighted time sequence, $\hat{\mathbf{h}}_w(k) = w(n, k)\hat{\mathbf{h}}(k)$ where $w(k, n)$ is a weighting function as in the linear algorithm. Then the two-sided z-transform is given by

$$\hat{\mathbf{H}}_w(z) = \sum_{k=-n}^{n} w(k, n)\hat{\mathbf{h}}(k)z^{-k} = \sum_{k=-n}^{n} \hat{\mathbf{h}}_w(k)z^{-k}. \tag{43}$$

The choice of windowing function has significant effects on the rates of convergence and on the bound on the worst-case identification error. A trade-off appears to exist between convergence rate (as n gets larger) and the worst case error due to noise (Gu and Khargonekar, 1992a). For example, the triangular window has a worst-case error convergence rate on the order of n (Gu and Khargonekar, 1990); the cosine window goes as order of n^2, but at the cost of a larger worst-case noise error. Similarly, the one-sided boxcar window (*i.e.*, $w(k, n)$ is 1 in $0 \le k \le n$ and zero elsewhere) has exponential convergence but has divergent worst case noise error (order of $\log n$) (Parker and Bitmead, 1987a; Gu and Khargonekar, 1992a). Noting this, Gu and Khargonekar (1992a) proposed using a trapezoidal windowing function whose convergence rate is between the triangular and one-sided boxcar, depending on the choice of the parameter m; the upper bound on the worst-case nonlinear identification error for a SISO system with a trapezoidal window is

$$\inf_{\substack{\text{admissible } \hat{F} \\ |\eta_k| \le \varepsilon}} \left\| H - \hat{F} \right\|_\infty \le \frac{2Mr}{1-r}\left[r^{N-1} + r^{N-n+m-1} + 2r^{2m}\right] + \left[\frac{2(n+m)}{n-m}\right]\varepsilon \tag{44}$$

where $r < 1$ is an upper bound on the relative stability and M is an upper bound on the steady state gain of H (Friedman and Khargonekar, 1995b).

The nonlinear algorithm, then, requires finding an $\hat{\mathbf{F}}(z)$ such that $\left\| \hat{\mathbf{H}}_w - \hat{\mathbf{F}} \right\|_\infty$ is at its infimum. $\hat{\mathbf{H}}_w(z)$ is a mixed causal/anticausal function (because positive powers of z imply prediction); a completely causal $\hat{\mathbf{F}}$ is desired. Several theorems will be useful in solving this problem.

Theorem 1: Given a causal $\hat{\underset{\sim}{\mathbf{H}}}(z)$, the anticausal $\hat{\underset{\sim}{\mathbf{F}}}(z)$ that infimizes $\left\| \hat{\underset{\sim}{\mathbf{H}}} - \hat{\underset{\sim}{\mathbf{F}}} \right\|_\infty$ is given by the solution to the *Nehari* problem (*e.g.*, Nehari, 1957; Adamjan, Arov, and Krein, 1971; Young, 1988; Dahleh and Diaz-Bobillo, 1995) as follows.

If $\hat{\underset{\sim}{\mathbf{H}}}$ has state space description

$$\hat{\underset{\sim}{\mathbf{H}}} = \left[\begin{array}{c|c} \mathbf{A} & \mathbf{B} \\ \hline \mathbf{C} & \mathbf{D} \end{array}\right], \quad \text{or equivalently,} \quad \begin{aligned} \mathbf{q}(k+1) &= \mathbf{A}\mathbf{q}(k) + \mathbf{B}\mathbf{u}(k) \\ \mathbf{y}(k) &= \mathbf{C}\mathbf{q}(k) + \mathbf{D}\mathbf{u}(k) \end{aligned}, \tag{45}$$

then the controllability and observability grammians, $\mathbf{P}$ and $\mathbf{Q}$, respectively, are the solutions to the Lyapunov equations

$$\mathbf{A}\mathbf{P}\mathbf{A}^{\mathrm{T}} + \mathbf{B}\mathbf{B}^{\mathrm{T}} = \mathbf{P} \quad \text{and} \quad \mathbf{A}^{\mathrm{T}}\mathbf{Q}\mathbf{A} + \mathbf{C}^{\mathrm{T}}\mathbf{C} = \mathbf{Q}. \tag{46}$$

Let the j^{th} right eigenvector of $\mathbf{PQ}$ be denoted by $\mathbf{x}_j$ with associated eigenvalues σ_j^2 (*i.e.*, $\mathbf{PQ}\mathbf{x}_j = \sigma_j^2\mathbf{x}_j$), where the indices are ordered such that $\sigma_1^2 \ge \sigma_2^2 \ge \ldots \ge \sigma_n^2$. Let $\mathbf{y}_j$ be defined by $\mathbf{y}_j = \mathbf{Q}\mathbf{x}_j/\sigma_j$, so that $\mathbf{x}_j$ and $\mathbf{y}_j$ will satisfy

$$\mathbf{P}\mathbf{y}_j = \sigma_j\mathbf{x}_j, \quad \mathbf{Q}\mathbf{x}_j = \sigma_j\mathbf{y}_j, \quad \text{and} \quad \mathbf{Q}\mathbf{P}\mathbf{y}_j = \sigma_j^2\mathbf{y}_j. \tag{47}$$

Then the *Schmidt pair*, $\mathbf{w}_j$ and $\mathbf{v}_j$, associated with σ_j can be expressed as

$$\begin{aligned}
\mathbf{w}_j(k) &= \mathbf{C}\mathbf{A}^k\mathbf{x}_j, \quad k \ge 0 & \mathbf{v}_j(k) &= \mathbf{B}^{\mathrm{T}}(\mathbf{A}^{\mathrm{T}})^{-(k+1)}\mathbf{y}_j, \quad k < 0 \\
& & \mathbf{V}_j(z) &= \mathbf{B}^{\mathrm{T}}\mathbf{y}_j z + \mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{y}_j z^2 + \mathbf{B}^{\mathrm{T}}(\mathbf{A}^{\mathrm{T}})^2\mathbf{y}_j z^3 + \ldots \\
\mathbf{W}_j(z) &= \mathbf{C}\mathbf{x}_j + \mathbf{C}\mathbf{A}\mathbf{x}_j z^{-1} + \mathbf{C}\mathbf{A}^2\mathbf{x}_j z^{-2} + \ldots & \underset{\sim}{\mathbf{V}}_j(z) &= \mathbf{V}_j(1/z) = \mathbf{B}^{\mathrm{T}}\mathbf{y}_j z^{-1} + \mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{y}_j z^{-2} + \ldots \\
&= \mathbf{C}\mathbf{A}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{x}_j + \mathbf{C}\mathbf{x}_j & &= \mathbf{B}^{\mathrm{T}}(z\mathbf{I} - \mathbf{A}^{\mathrm{T}})^{-1}\mathbf{y}_j \\
&= \left[\begin{array}{c|c} \mathbf{A} & \mathbf{x}_j \\ \hline \mathbf{C}\mathbf{A} & \mathbf{C}\mathbf{x}_j \end{array}\right] & &= \left[\begin{array}{c|c} \mathbf{A}^{\mathrm{T}} & \mathbf{y}_j \\ \hline \mathbf{B}^{\mathrm{T}} & 0 \end{array}\right]
\end{aligned} \tag{48}$$

Finally, the solution to the *Nehari* problem is

$$\inf_{\text{anticausal } \hat{\underline{F}}} \left\| \hat{\underline{H}} - \hat{\underline{F}} \right\|_\infty = \sigma_1 \qquad [\hat{\underline{H}}(z) - \hat{\underline{F}}(z)]\, \mathbf{V}_1(z) = \sigma_1 \mathbf{W}_1(z). \tag{49}$$

For SIMO systems, the optimal $\hat{\underline{F}}(z)$ has a unique solution given by

$$\hat{\underline{F}}(z) = \hat{\underline{H}}(z) - \sigma_1 \frac{\mathbf{W}_1(z)}{V_1(z)}. \tag{50}$$

Proof of this theorem is found in Dahleh and Diaz-Bobillo (1995). ■

Theorem 2: Given an anticausal $\hat{\mathbf{H}}_-$, the infimal value of $\left\| \hat{\mathbf{H}}_- - \hat{\underline{F}} \right\|_\infty$ over all causal $\hat{\underline{F}}$ and the argument that infimizes it are

$$\inf_{\text{anticausal } \hat{\underline{F}}} \left\| \hat{\mathbf{H}}_- - \hat{\underline{F}} \right\|_\infty = \sigma_1 \qquad [\hat{\mathbf{H}}_-(z) - \hat{\underline{F}}(z)]\, \mathbf{V}_1(1/z) = \sigma_1 \mathbf{W}_1(1/z), \tag{51}$$

and for SIMO systems, the infimizing $\hat{\underline{F}}(z)$ is given uniquely by

$$\hat{\underline{F}}(z) = \hat{\mathbf{H}}_-(z) - \sigma_1 \frac{\mathbf{W}_1(1/z)}{V_1(1/z)}. \tag{52}$$

The proof merely requires letting $\hat{\underline{H}}(1/z) = \hat{\mathbf{H}}_-(z)$ and $\hat{\underline{F}}(1/z) = \hat{\underline{F}}(z)$, such that $\hat{\underline{H}}$ is causal and $\hat{\underline{F}}$ is anticausal, and applying Theorem 1. ■

Theorem 3: Given a mixed causal/anticausal $\hat{\mathbf{H}}_w$, the infimal value of $\left\| \hat{\mathbf{H}}_w - \hat{\mathbf{F}} \right\|_\infty$ over all causal $\hat{\mathbf{F}}$ and the argument that infimizes it are

$$\inf_{\text{causal } \hat{\mathbf{F}}} \left\| \hat{\mathbf{H}}_w - \hat{\mathbf{F}} \right\|_\infty = \sigma_1 \qquad [\hat{\mathbf{H}}_w(z) - \hat{\mathbf{F}}(z)]\, \underline{\mathbf{V}}_1(z) = \sigma_1 \mathbf{W}_1(1/z) \tag{53}$$

The proof is as follows. Let $\hat{\mathbf{H}}_w(z) = \hat{\mathbf{H}}_+(z) + \hat{\mathbf{H}}_-(z)$, separating $\hat{\mathbf{H}}_w$ into its purely causal and purely anticausal parts, $\hat{\mathbf{H}}_+$ and $\hat{\mathbf{H}}_-$, respectively,

$$\begin{aligned} \hat{\mathbf{H}}_+(z) &= \sum_{k=0}^{\infty} \hat{\mathbf{h}}_w(k) z^{-k} = \hat{\mathbf{h}}_w(0) + \hat{\mathbf{h}}_w(1) z^{-1} + \hat{\mathbf{h}}_w(2) z^{-2} + \dots \\ \hat{\mathbf{H}}_-(z) &= \sum_{k=-\infty}^{-1} \hat{\mathbf{h}}_w(k) z^{-k} = \hat{\mathbf{h}}_w(-1) z^{1} + \hat{\mathbf{h}}_w(-2) z^{2} + \dots \end{aligned} \tag{54}$$

Further let $\hat{\underline{F}} = \hat{\mathbf{F}} - \hat{\mathbf{H}}_+$. Theorem 2 is applied to find the causal $\hat{\underline{F}}$ nearest the anticausal $\hat{\mathbf{H}}_-$. But since $\hat{\mathbf{H}}_w - \hat{\mathbf{F}} = \hat{\mathbf{H}}_- - \hat{\underline{F}}$,

$$\inf_{\text{causal } \hat{\mathbf{F}}} \left\| \hat{\mathbf{H}}_w - \hat{\mathbf{F}} \right\|_\infty = \sigma_1 \qquad [\hat{\mathbf{H}}_w(z) - \hat{\mathbf{F}}(z)]\, \underset{\sim}{\mathbf{V}}_1(z) = \sigma_1 \mathbf{W}_1(1/z), \tag{55}$$

and for SIMO systems, a unique solution exists

$$\hat{\mathbf{F}}(z) = \hat{\mathbf{H}}_w(z) - \sigma_1 \frac{\mathbf{W}_1(1/z)}{\underset{\sim}{V}_1(z)} \tag{56}$$

where $\mathbf{W}_1$ and V_1 are the Schmidt pair given in Theorem 1. ■

Theorem 4: The system $\underset{\sim}{\hat{\mathbf{H}}}$ for a SIMO problem is such that

(*a*) the controllability grammian $\mathbf{P}$ is the identity matrix,

(*b*) the observability grammian is given by

$$\mathbf{Q} = \sum_{r=1}^{n_o} \mathbf{\Lambda}_r^2, \tag{57}$$

where n_o is the number of outputs and $\mathbf{\Lambda}_r$ denotes the Hankel matrix associated with the anticausal part of the r^{th} pulse response $\hat{h}_{w_r}(-k)$; *i.e.*,

$$\mathbf{\Lambda}_r = \begin{bmatrix} \hat{h}_{w_r}(-1) & \hat{h}_{w_r}(-2) & \cdots & \hat{h}_{w_r}(1-n) & \hat{h}_{w_r}(-n) \\ \hat{h}_{w_r}(-2) & \hat{h}_{w_r}(-3) & \cdots & \hat{h}_{w_r}(-n) & \\ \vdots & \vdots & \iddots & & \\ \hat{h}_{w_r}(1-n) & \hat{h}_{w_r}(-n) & & \mathbf{0} & \\ \hat{h}_{w_r}(-n) & & & & \end{bmatrix}, \tag{58}$$

(*c*) σ_1^2 is the largest eigenvalue value of $\mathbf{Q}$,

(*d*) $\mathbf{y}_1 = \sigma_1 \mathbf{x}_1$, and

(*e*) the system estimate $\hat{\mathbf{F}}$ reduces to the n_o equations

$$\hat{F}_r(z) = \frac{\lfloor z^{2n-1} \quad \cdots \quad z \quad 1 \rfloor \underset{\sim}{\mathbf{\Lambda}}_r \mathbf{x}_1}{\lfloor z^{2n-1} \quad \cdots \quad z^{n+1} \quad z^n \rfloor \mathbf{x}_1} \tag{59}$$

where $\underset{\sim}{\mathbf{\Lambda}}_r$ is a function of the r^{th} pulse response $\hat{h}_{w_r}(k)$ such that its elements are given by

$$\underset{\sim}{\lambda}_{r_{jk}} = \begin{cases} \hat{h}_{w_r}(j-k), & j-k \le 0 \\ 0, & j-k > 0 \end{cases} . \tag{60}$$

The character of the state space form of $\hat{\underset{\sim}{\mathbf{H}}}$ for a SIMO system is:

$$\mathbf{A} = \left[\begin{array}{c|c} \mathbf{0}_{1\times(n-1)} & 0 \\ \hline \mathbf{I}_{(n-1)\times(n-1)} & \mathbf{0}_{(n-1)\times 1} \end{array}\right] \qquad \mathbf{B} = \left\{\begin{array}{c} 1 \\ \hline \mathbf{0}_{(n-1)\times 1} \end{array}\right\} \tag{61}$$

$$\mathbf{C} = \begin{bmatrix} \hat{\mathbf{h}}_w(-1) & \hat{\mathbf{h}}_w(-2) & \cdots & \hat{\mathbf{h}}_w(-n) \end{bmatrix} \qquad \mathbf{D} = 0$$

Because of this simple structure, the Lyapunov equations can be symbolically solved. The Lyapunov equation for $\mathbf{P}$ is $\mathbf{APA}^{\mathrm{T}} + \mathbf{BB}^{\mathrm{T}} = \mathbf{P}$. The terms on the left-hand side can be written as:

$$\mathbf{APA}^{\mathrm{T}} = \left[\begin{array}{c|c} \mathbf{0}_{1\times(n-1)} & 0 \\ \hline \mathbf{I}_{(n-1)\times(n-1)} & \mathbf{0}_{(n-1)\times 1} \end{array}\right] \left[\begin{array}{c|c} \mathbf{P}_a & \mathbf{P}_b \\ \hline \mathbf{P}_c & p_{nn} \end{array}\right] \left[\begin{array}{c|c} \mathbf{0}_{(n-1)\times 1} & \mathbf{I}_{(n-1)\times(n-1)} \\ \hline 0 & \mathbf{0}_{1\times(n-1)} \end{array}\right] \tag{62}$$

$$= \left[\begin{array}{c|c} 0 & \mathbf{0}_{1\times(n-1)} \\ \hline \mathbf{0}_{(n-1)\times 1} & \mathbf{P}_a \end{array}\right]$$

$$\mathbf{BB}^{\mathrm{T}} = \left\{\begin{array}{c} 1 \\ \hline \mathbf{0}_{(n-1)\times 1} \end{array}\right\} \left[\begin{array}{c|c} 1 & \mathbf{0}_{1\times(n-1)} \end{array}\right] = \left[\begin{array}{c|c} 1 & \mathbf{0}_{1\times(n-1)} \\ \hline \mathbf{0}_{(n-1)\times 1} & \mathbf{0}_{(n-1)\times(n-1)} \end{array}\right] \tag{63}$$

$$\mathbf{APA}^{\mathrm{T}} + \mathbf{BB}^{\mathrm{T}} = \left[\begin{array}{c|c} 1 & \mathbf{0}_{1\times(n-1)} \\ \hline \mathbf{0}_{(n-1)\times 1} & \mathbf{P}_a \end{array}\right] = \mathbf{P} \tag{64}$$

Therefore, $p_{11} = 1$, $p_{1j} = p_{j1} = 0$, and $p_{jk} = p_{(j-1)(k-1)}$ for $j = 2, \ldots, n$ and $k = 2, \ldots, n$. So, result (*a*) is

$$p_{jk} = \delta_{jk} = \begin{cases} 1, & j = k \\ 0, & j \neq k \end{cases} \qquad \Rightarrow \qquad \mathbf{P} = \mathbf{I}_{n \times n} \tag{65}$$

A similar simplification of the other Lyapunov equation, $\mathbf{A}^T\mathbf{Q}\mathbf{A} + \mathbf{C}^T\mathbf{C} = \mathbf{Q}$, gives result (*b*)

$$\begin{aligned} q_{jk} &= \begin{cases} \hat{\mathbf{h}}_w^T(-j)\hat{\mathbf{h}}_w(-k), & j = n \text{ or } k = n \\ q_{(j+1)(k+1)} + \hat{\mathbf{h}}_w^T(-j)\hat{\mathbf{h}}_w(-k), & \text{otherwise} \end{cases} \\ &= \sum_{m = \max(j,k)}^{n} \hat{\mathbf{h}}_w^T(-m)\hat{\mathbf{h}}_w(|j-k| - m) \qquad \Rightarrow \qquad \mathbf{Q} = \sum_{r=1}^{n_o} \mathbf{\Lambda}_r^2. \end{aligned} \tag{66}$$

Since σ_1^2 is the largest eigenvalue of $\mathbf{PQ}$ and $\mathbf{P}$ is the identity matrix, result (*c*) is immediate. Similarly, because $\mathbf{P}$ is the identity matrix, (47) simplifies to result (*d*) $\mathbf{y}_1 = \sigma_1 \mathbf{x}_1$.

The final result (*e*) is found by substituting the definitions of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ in (61) into the definition of the Schmidt pair (48) and simplifying

$$\begin{aligned} W_{1_r}(1/z) &= \lfloor \hat{h}_{w_r}(-1) \;\; \hat{h}_{w_r}(-2) \;\; \ldots \;\; \hat{h}_{w_r}(-n) \rfloor \begin{bmatrix} 1 & & & \mathbf{0} \\ z & 1 & & \\ z^2 & z & 1 & \\ \vdots & \ddots & \ddots & \ddots \\ z^{n-1} & \cdots & z^2 & z \;\; 1 \end{bmatrix} \mathbf{x}_1 \\ &= \lfloor 1 \;\; z \;\; \ldots \;\; z^{n-1} \rfloor \mathbf{\Lambda}_r \mathbf{x}_1 \\ \underset{\sim}{V}_1(z) &= \lfloor z^{-1} \;\; z^{-2} \;\; \ldots \;\; z^{-n} \rfloor \mathbf{y}_1 = \sigma_1 \lfloor z^{-1} \;\; z^{-2} \;\; \ldots \;\; z^{-n} \rfloor \mathbf{x}_1 \end{aligned} \tag{67}$$

Substituting into (56),

$$\begin{aligned} \hat{F}_r(z) &= [\hat{H}_{w_r}(z)\underset{\sim}{V}_1(z) - \sigma_1 W_{1_r}(1/z)] / \underset{\sim}{V}_1(z) \\ &= \frac{\hat{H}_{w_r}(z)\lfloor z^{-1} \;\; z^{-2} \;\; \ldots \;\; z^{-n} \rfloor \mathbf{x}_1 - \lfloor 1 \;\; z \;\; \ldots \;\; z^{n-1} \rfloor \mathbf{\Lambda}_r \mathbf{x}_1}{\lfloor z^{-1} \;\; z^{-2} \;\; \ldots \;\; z^{-n} \rfloor \mathbf{x}_1} \\ &= \frac{\left[\hat{H}_{w_r}(z)\lfloor z^{-1} \;\; z^{-2} \;\; \ldots \;\; z^{-n} \rfloor - \lfloor 1 \;\; z \;\; \ldots \;\; z^{n-1} \rfloor \mathbf{\Lambda}_r \right] \mathbf{x}_1}{\lfloor z^{-1} \;\; z^{-2} \;\; \ldots \;\; z^{-n} \rfloor \mathbf{x}_1} \end{aligned} \tag{68}$$

But the first term in the numerator can be expanded as

$$\hat{H}_{w_r}(z)\begin{Bmatrix} z^{-1} \\ z^{-2} \\ \vdots \\ z^{-n} \end{Bmatrix}^{\mathrm{T}} = \begin{Bmatrix} \hat{h}_{w_r}(0)z^{-1} + \ldots + \hat{h}_{w_r}(n)z^{-n-1} \\ \hat{h}_{w_r}(-1)z^{-1} + \quad \ldots \quad + \hat{h}_{w_r}(n)z^{-n-2} \\ \vdots \qquad\qquad\qquad\qquad \ddots \\ \hat{h}_{w_r}(1-n)z^{-1} + \qquad \ldots \qquad + \hat{h}_{w_r}(n)z^{-2n} \end{Bmatrix} + \begin{Bmatrix} \hat{h}_{w_r}(-n)z^{n-1} + \ldots + \hat{h}_{w_r}(-1) \\ \vdots \qquad\qquad\qquad \iddots \\ \hat{h}_{w_r}(-n) \end{Bmatrix}$$

$$= \begin{Bmatrix} z^{-1} \\ z^{-2} \\ \vdots \\ z^{-2n} \end{Bmatrix}^{\mathrm{T}} \begin{bmatrix} \hat{h}_{w_r}(0) & \hat{h}_{w_r}(-1) & \cdots & \hat{h}_{w_r}(1-n) \\ \hat{h}_{w_r}(1) & \hat{h}_{w_r}(0) & \cdots & \hat{h}_{w_r}(2-n) \\ \vdots & & & \\ \hat{h}_{w_r}(n-1) & \vdots & & \vdots \\ \hat{h}_{w_r}(n) & \hat{h}_{w_r}(n-1) & & \\ & \hat{h}_{w_r}(n) & \ddots & \\ & & \ddots & \hat{h}_{w_r}(n-1) \\ & \mathbf{0} & & \hat{h}_{w_r}(n) \end{bmatrix} + \begin{Bmatrix} 1 \\ z^1 \\ \vdots \\ z^{n-1} \end{Bmatrix}^{\mathrm{T}} \begin{bmatrix} \hat{h}_{w_r}(-1) & \cdots & \hat{h}_{w_r}(-n) \\ \vdots & \iddots & \\ \hat{h}_{w_r}(-n) & \mathbf{0} & \end{bmatrix} \tag{69}$$

$$= \lfloor z^{-1} \quad z^{-2} \quad \ldots \quad z^{-2n} \rfloor \underset{\sim}{\Lambda}_r + \lfloor 1 \quad z \quad \ldots \quad z^{n-1} \rfloor \Lambda_r$$

Thus the model for the r^{th} output can be simplified to

$$\hat{F}_r(z) = \frac{[\lfloor z^{-1} \quad z^{-2} \quad \ldots \quad z^{-2n} \rfloor \underset{\sim}{\Lambda}_r + \lfloor 1 \quad z \quad \ldots \quad z^{n-1} \rfloor \Lambda_r - \lfloor 1 \quad z \quad \ldots \quad z^{n-1} \rfloor \Lambda_r] \mathbf{x}_1}{\lfloor z^{-1} \quad z^{-2} \quad \ldots \quad z^{-n} \rfloor \mathbf{x}_1}$$

$$= \frac{\lfloor z^{-1} \quad z^{-2} \quad \ldots \quad z^{-2n} \rfloor \underset{\sim}{\Lambda}_r \mathbf{x}_1}{\lfloor z^{-1} \quad z^{-2} \quad \ldots \quad z^{-n} \rfloor \mathbf{x}_1} \tag{70}$$

$$= \frac{\lfloor z^{2n-1} \quad \ldots \quad z \quad 1 \rfloor \underset{\sim}{\Lambda}_r \mathbf{x}_1}{\lfloor z^{2n-1} \quad \ldots \quad z^{n+1} \quad z^n \rfloor \mathbf{x}_1}$$

which proves result (*e*). ■

In summary, to compute the estimate $\hat{\mathbf{F}}$ of the SIMO system $\mathbf{H}$ and the estimated bound on the model error:

1. Construct the Hankel matrices Λ_r from the anticausal part of the weighted pulse responses

$$\lambda_{r_{jk}} = \begin{cases} \hat{h}_{w_r}(1-j-k), & j+k-1 \le n \\ 0, & j+k-1 > n \end{cases} \tag{71}$$

2. Find the largest eigenvalue σ_1^2 and the corresponding eigenvector $\mathbf{x}_1$ of the sum of the squares of the anticausal Hankel matrices

$$\left(\sum_{r=1}^{n_o} \Lambda_r^2 \right) \mathbf{x}_1 = \sigma_1^2 \mathbf{x}_1 \tag{72}$$

3. Form the $\underset{\sim}{\Lambda}_r$ matrices

$$\underset{\sim}{\lambda}_{r_{jk}} = \begin{cases} \hat{h}_{w_r}(j-k), & j-k \le 0 \\ 0, & j-k > 0 \end{cases} \tag{73}$$

4. Then the estimated model $\hat{\mathbf{F}}$ and the estimated bound on the model error are

$$\hat{\mathbf{F}}(z) = \frac{1}{\lfloor z^{2n-1} \quad \dots \quad z^{n+1} \quad z^n \rfloor \mathbf{x}_1} \begin{Bmatrix} \lfloor z^{2n-1} & \dots & z & 1 \rfloor \underset{\sim}{\Lambda}_1 \\ \lfloor z^{2n-1} & \dots & z & 1 \rfloor \underset{\sim}{\Lambda}_2 \\ & \vdots & & \\ \lfloor z^{2n-1} & \dots & z & 1 \rfloor \underset{\sim}{\Lambda}_{n_o} \end{Bmatrix} \mathbf{x}_1 \tag{74}$$

$$\left\| \hat{\mathbf{H}} - \hat{\mathbf{F}} \right\|_\infty = \sigma_1 \tag{75}$$

4.1.3 Example I of the H_∞-based Identification Algorithms

Gu and Khargonekar (1992a, 1992b) give some examples of the untuned linear and nonlinear algorithms; one such example (Gu and Khargonekar, 1992b), repeated here, is the attempt to identify the system

$$H(z) = \frac{3(z^2+1)}{5z^2+2z+1} \tag{76}$$

with noise bound $|\eta_k| \le \varepsilon = 0.1$. The noise is generated by $\eta_k = \varepsilon e^{i\theta_k}$, where θ_k is a uniformly distributed random variable on the interval $[0, 2\pi)$. N is chosen to be 512. A triangular window is used, with n taking on the values 5, 10, 20, 40, and 80. Both the linear and nonlinear algorithms were executed using MATLAB® on the UIUC Engineering Workstations.

Figure 11 shows the pulse response of the system with noise and without. The pulse response of the exact system and the identified models are shown in Fig. 12. Only the lowest-order model ($n = 5$) shows significant deviation from the actual system response.

The identified transfer functions are plotted with the original system transfer function in Figs. 13 and 14 (magnitude and phase respectively; frequency is normalized such that the Nyquist frequency is normalized to 1). It is obvious that the larger the order of the identification (*i.e.*, the larger the value of n), the closer the approximation is to the original transfer function. The worst-case error, give in Table 1, seems to suggest that the converse would be true; in all cases examined

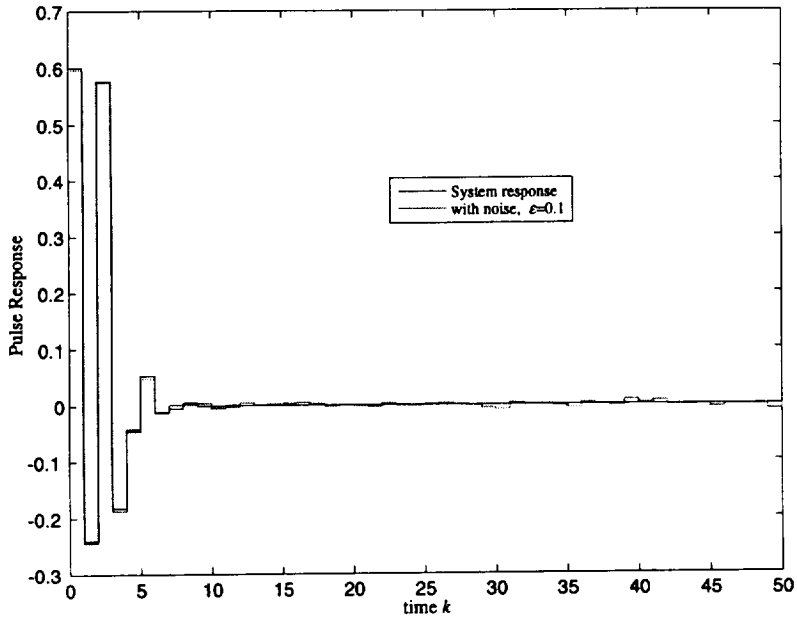


Figure 11: Response of system with and without noise.

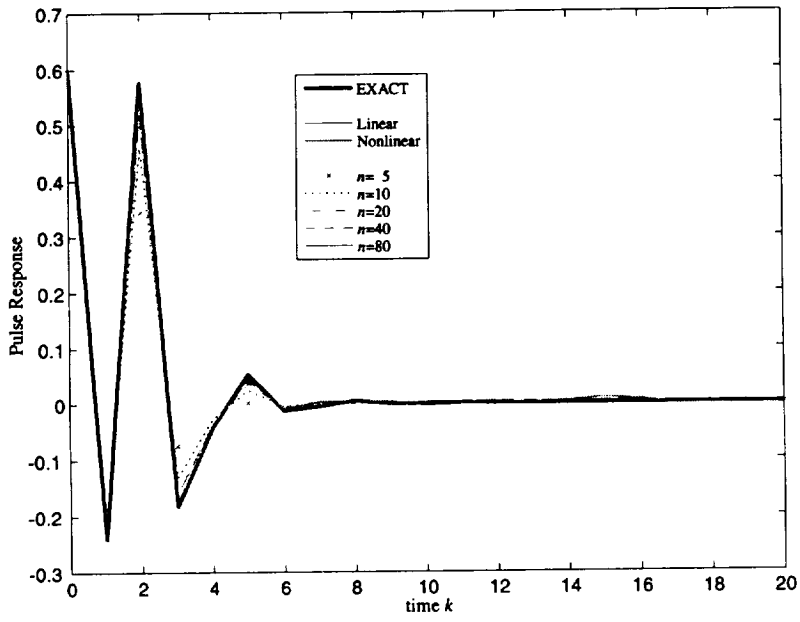


Figure 12: Pulse response of original system and identified models.

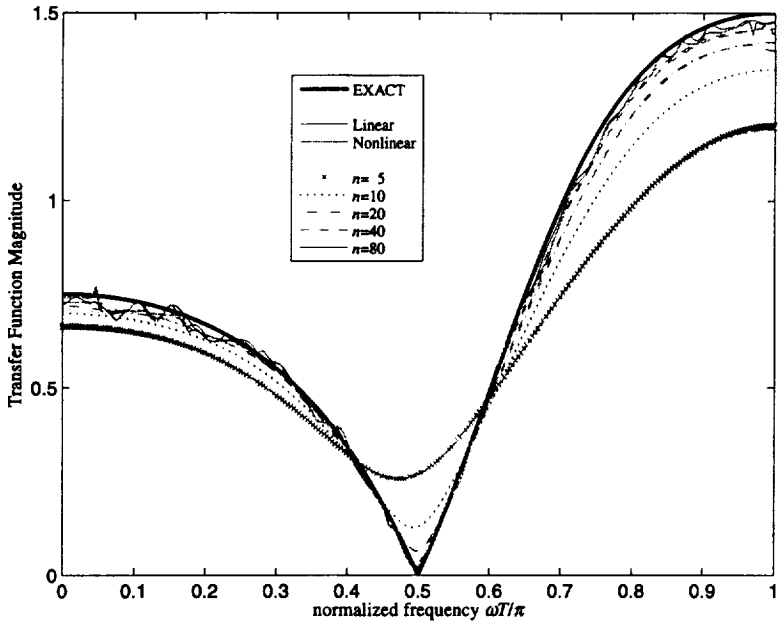


Figure 13: Magnitude of original and identified transfer functions.

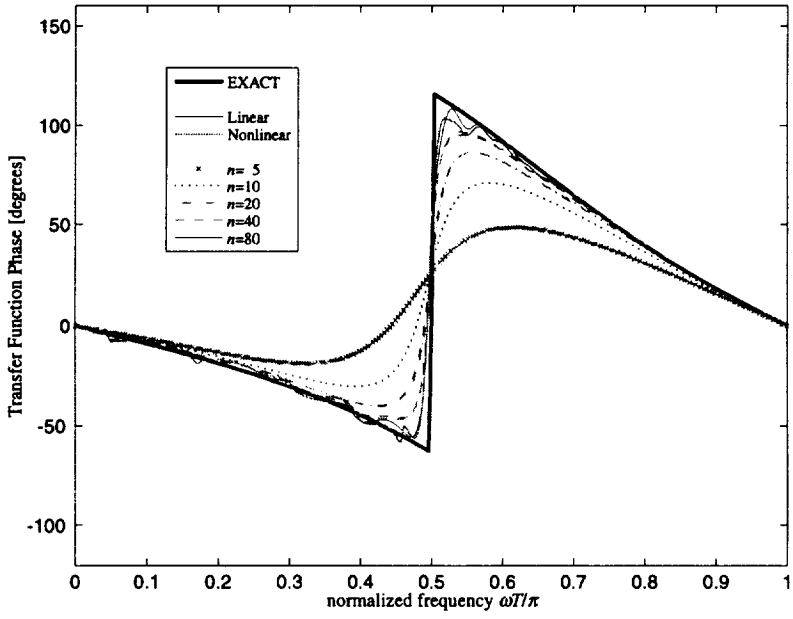


Figure 14: Phase of original and identified transfer functions.

model order n	*worst-case error* σ_1
5	0.0079059
10	0.011599
20	0.015108
40	0.019927
80	0.029021

Table 1: Worst-case error for the nonlinear identified models of Example I.

for this study, however, the actual error never went up with increasing model order (unless numerical difficulties in the eigenproblem corrupted the solution).

Due to the high order of the approximations, only the lowest-order identified polynomial models will be given here (the polynomial coefficients are accurate to 4 significant digits):

$$\hat{F}_{\substack{\text{linear} \\ n=5}}(z) = \frac{0.5961z^4 - 0.1951z^3 + 0.3449z^2 - 0.07495z - 0.009165}{z^4}$$

$$\hat{F}_{\substack{\text{nonlinear} \\ n=5}}(z) = \frac{0.596z^6 - 0.1349z^5 - 0.000366z^4 + 0.06508z^3 - 0.2053z^2 + 0.04018z + 0.005021}{z^6}$$

$$\hat{F}_{\substack{\text{linear} \\ n=10}}(z) = \frac{\begin{array}{l}[0.5961z^9 - 0.2195z^8 + 0.4599z^7 - 0.1312z^6 - 0.0275z^5 \\ \quad + 0.0241z^4 - 0.00449z^3 + 0.0003689z^2 + 0.00021z + 0.0003871\ ]\end{array}}{z^9} \tag{77}$$

$$\hat{F}_{\substack{\text{nonlinear} \\ n=10}}(z) = \frac{\begin{array}{l}[0.5959z^{17} - 0.1136z^{16} + 0.04711z^{15} - 0.009078z^{14} - 0.2901z^{13} + 0.0837z^{12} \\ \quad - 0.6885z^{11} + 0.07275z^{10} - 0.06215z^9 + 0.01491z^8 + 0.02792z^7 - 0.01151z^6 \\ \quad - 0.0003642z^5 + 0.001117z^4 - 0.0002158z^3 - 0.00004217z^2 + 0.0000193\ ]\end{array}}{\begin{array}{l}[z^{17} + 0.1704z^{16} - 0.6321z^{15} - 0.1568z^{14} + 0.02693z^{13} \\ \quad + 0.09749z^{12} - 0.1631z^{11} + 0.01218z^{10} + 0.04985z^9\ ]\end{array}}$$

One important observation from the transfer functions is that the highest order models (*i.e.*, those with $n = 80$) are oscillating, in both magnitude and phase, unlike the lower order models, though it is closer to the exact transfer function than those lower order models; the higher-order model may be trying to model the noise. Another observation is that the nonlinear algorithm does not appear to do much better than the linear algorithm. In fact, the difference is only noticeable in Figures 15 and 16, that show the transfer function error magnitude, *i.e.*, $|e(z)| = |H(z) - \hat{F}(z)|$. (Figure 16 is quite similar to that shown by Gu and Khargonekar (1992b); any difference is due to the fact that the noise is random.)

The error of the pulse response of the identified models is shown in Fig. 17; the peak response error is approximately $O(1/n)$. The difference between the linear and nonlinear results are barely apparent here. Part of the reason is that the pulse response dies out so quickly (and indeed is zero

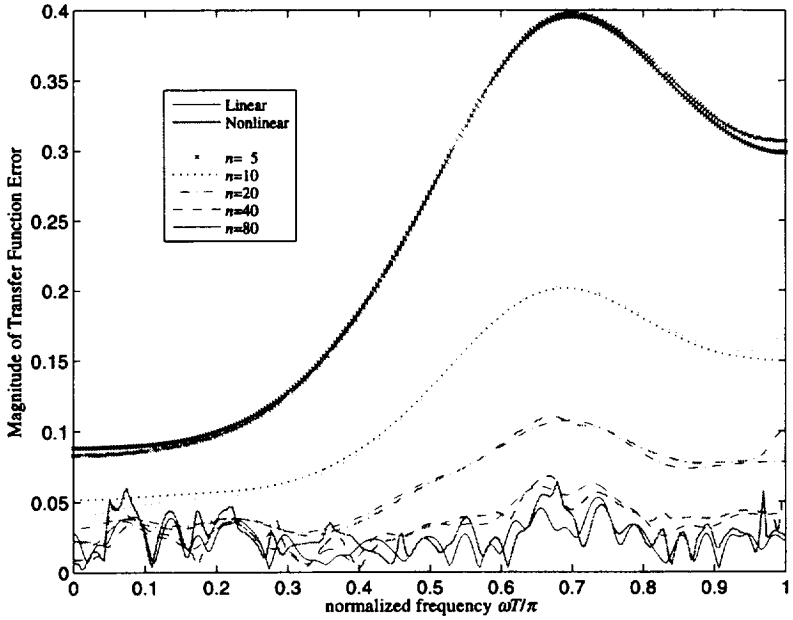


Figure 15: Magnitude of identified transfer function error.

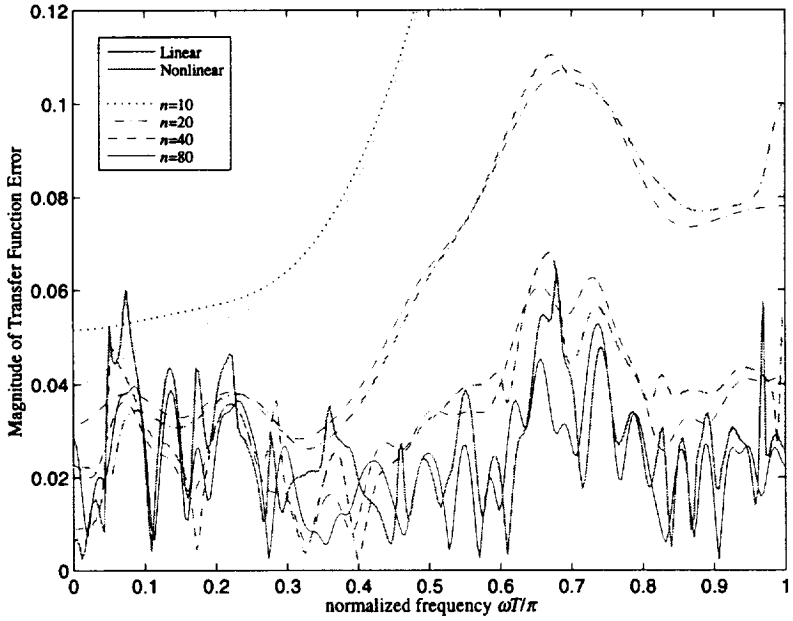


Figure 16: Magnitude of identified transfer function error.

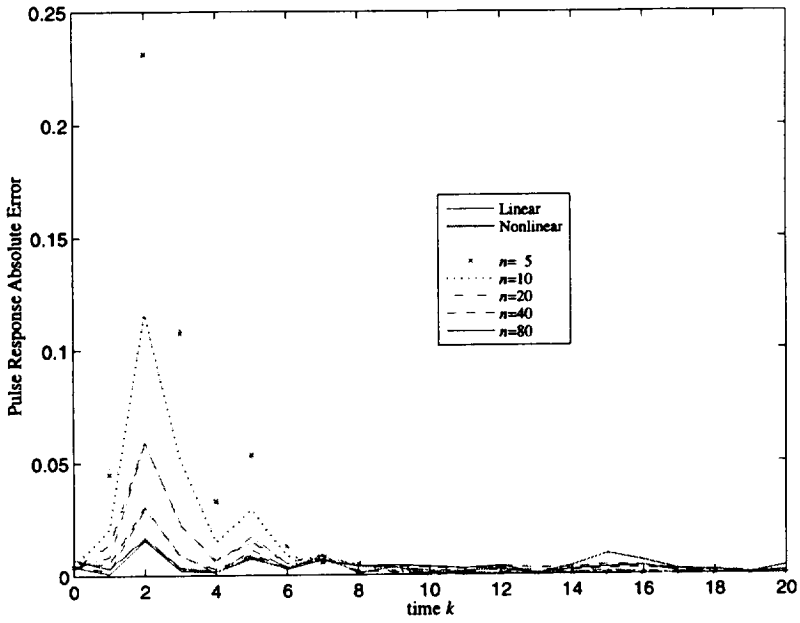


Figure 17: Absolute error in pulse responses of identified models.

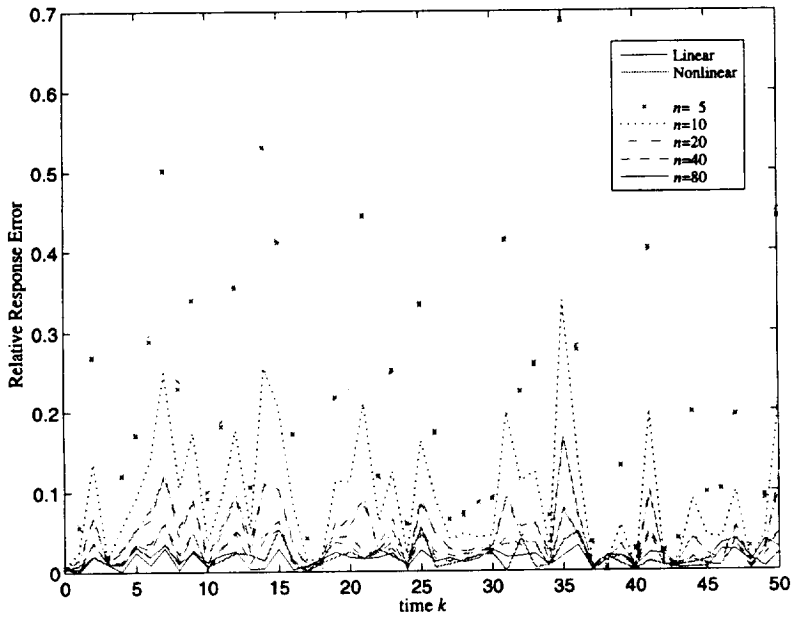


Figure 18: Error in response to Gaussian white noise input, relative to the RMS of the exact response.

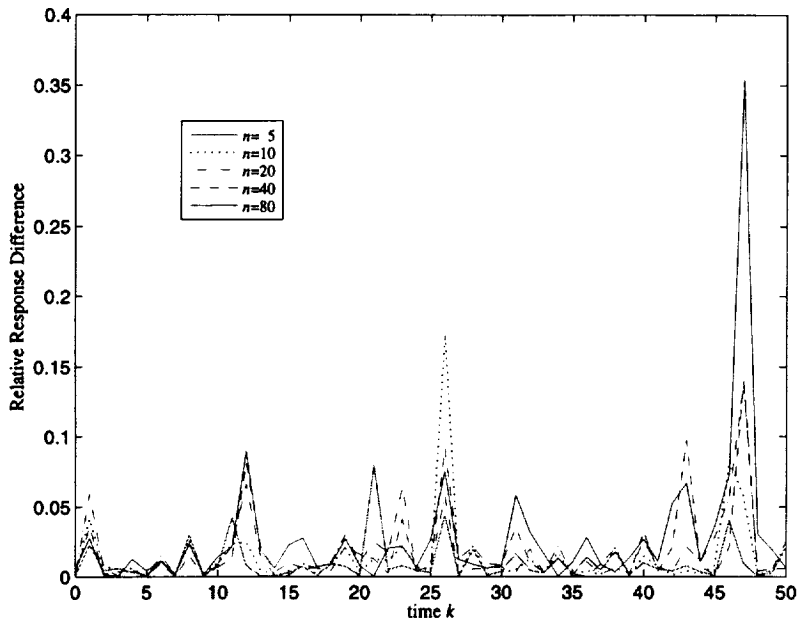


Figure 19: Relative difference between linear and nonlinear responses to Gaussian white noise input.

for linear approximations after time $n-1$). To examine a longer response, the exact system and the identified models are subjected to a Gaussian white noise input. The response error, relative to the root mean-square (RMS) of the exact response, is shown in Fig. 18. Again, the difference between the linear and nonlinear models are only apparent for the higher order models. This can be more easily seen in Fig. 19, which shows the relative response difference between corresponding linear and nonlinear models.

4.1.4 Example II of the H_∞-based Identification Algorithms

To evaluate the utility of H_∞-based identification for structural system identification, an additional example, using a structurally-based system, is beneficial. A six degree of freedom system, like that in Fig. 4, with all masses, damping coefficients, and spring stiffnesses the same, is simulated in discrete time; the system input is the force on the first mass, and the outputs are the displacements of the six masses.

The noise in the pulse responses is generated in the same manner as in the previous example, with the noise magnitude from each response being a constant $\varepsilon = 0.1$ magnitude in the frequency domain but of random phase. One of the six pulse responses, that of the third mass, is given in Fig. 20 with and without the noise. This system has, of course, six modes, but the damping varies significantly from mode to mode; the lowest frequency has 1.2% damping and the highest has 9.7%. Thus, the low frequencies dominate the long-term pulse response and the higher frequencies are only seen in the first portion of the response. This dominance can also be seen in the magnitude of the transfer functions of the exact system and to modal coordinates in Fig. 21. Since the noise has a flat frequency content, one would expect the identification of higher modes to be more error prone due to a lower signal-to-noise ratio.

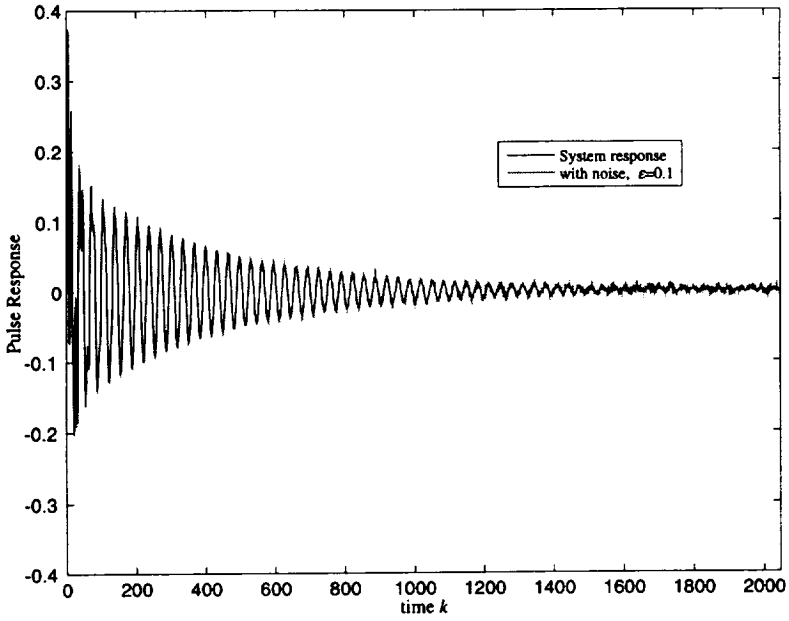


Figure 20: Pulse response of mass #3 of six degree of freedom system.

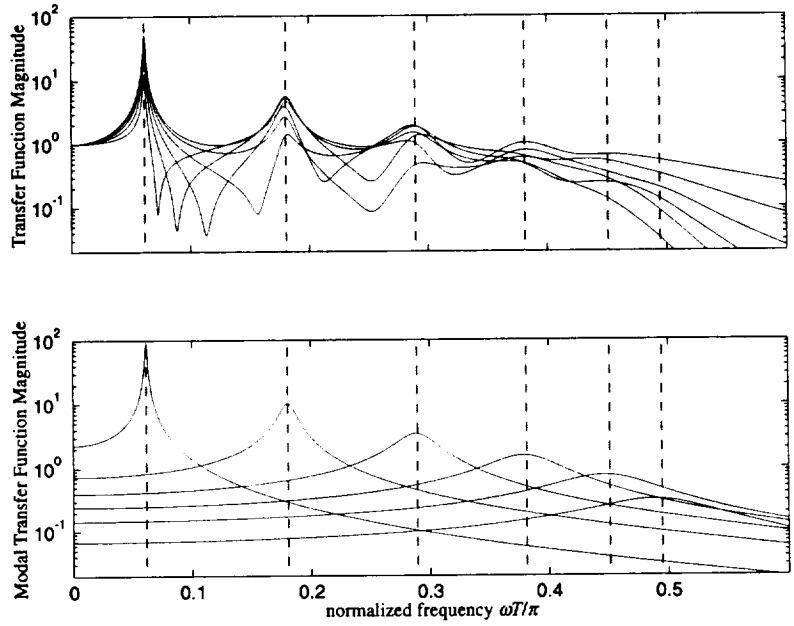


Figure 21: Transfer functions of the exact system (top) and to modal coordinates (bottom).

model order n	*worst-case error* σ_1
24	0.020505
60	0.035888
120	0.056611
240	0.10154
480	0.18838

Table 2: Worst-case error for the nonlinear identified models of Example II.

Both linear and nonlinear algorithms are used to identify this system. Their results are similar, so only the nonlinear results will be given here and the differences noted where appropriate. The worst-case error found by the nonlinear algorithms, shown in Table 1, increases with model order, as seen in the previous example. The actual error, however, was seen to be non-increasing with higher model order.

The magnitude and phase of the exact and nonlinear identified transfer functions to the first output are shown in Figs. 22 and 23. Decent estimates are found with a model order of $n = 120$; the strong peak of the first mode, required to accurately determine its damping, is not closely approximated unless an even higher order is used. The error at magnitude peaks and valleys is even more obvious in the transfer function to the third output in Fig. 24.

The error in the transfer functions demonstrates that the identified models have difficulty in obtaining accuracy with error magnitude less than the noise magnitude ε. Figure 25 shows the error magnitude of the nonlinear identified models for the first output. The peak error is seen to be non-increasing as model order increases, but the error is rarely much below the noise magnitude of 0.1.

As noted above, the difference between the linear and nonlinear identified models would be hard to see in a graph of the transfer function magnitudes. They do differ, but only for high model orders is the magnitude of that difference significant relative to the error in the transfer functions. Figure 26 shows the magnitude of the difference between the linear and nonlinear identified transfer functions for the third output. Only the difference for $n \geq 240$ is not small in comparison with the noise magnitude and the magnitude of the error of the identified models.

The H_∞-based identification does function satisfactorily for this structural identification problem.

4.1.5 Evaluation of the H_∞-based Identification Algorithms

The results of the H_∞-based identification are generally good, as was seen in the above examples. Application to real-world systems has also demonstrated the usefulness of H_∞-based identification in general (*e.g.*, Gu and Khargonekar, 1993; Friedman and Khargonekar, 1995a, 1995b, which identified the ATB1000 testbed at the U.S. Army Automation and Robotics Lab and the Advanced Reconfigurable Control (ARC) testbed, a 6-bay truss structure, at the Jet Propulsion Lab; Friedman, 1996).

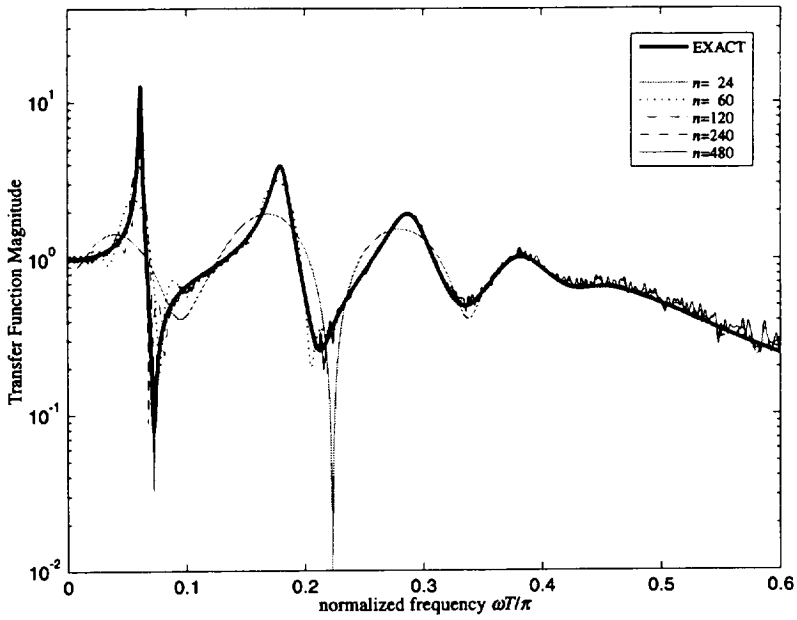


Figure 22: Magnitude of transfer function #1 of the exact system and the nonlinear identified models.

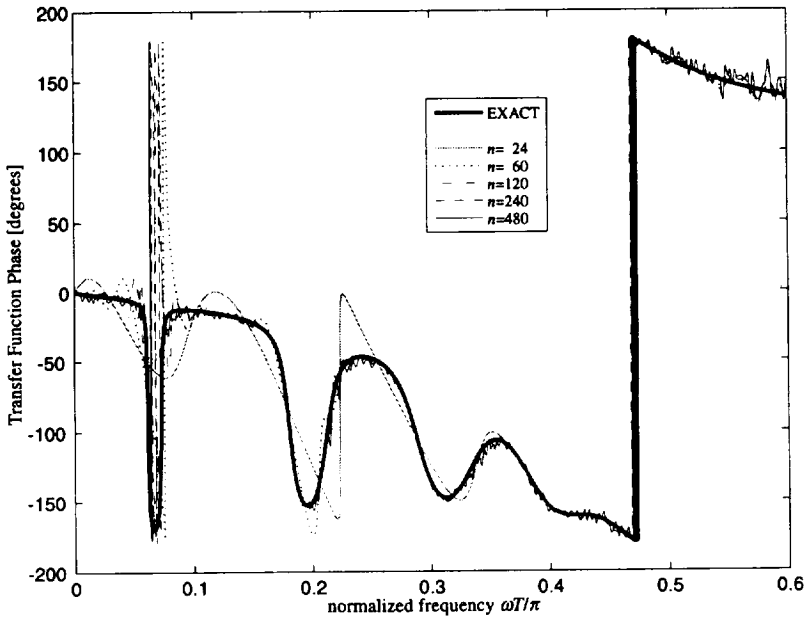


Figure 23: Phase of transfer function #1 of the exact system and the nonlinear identified models.

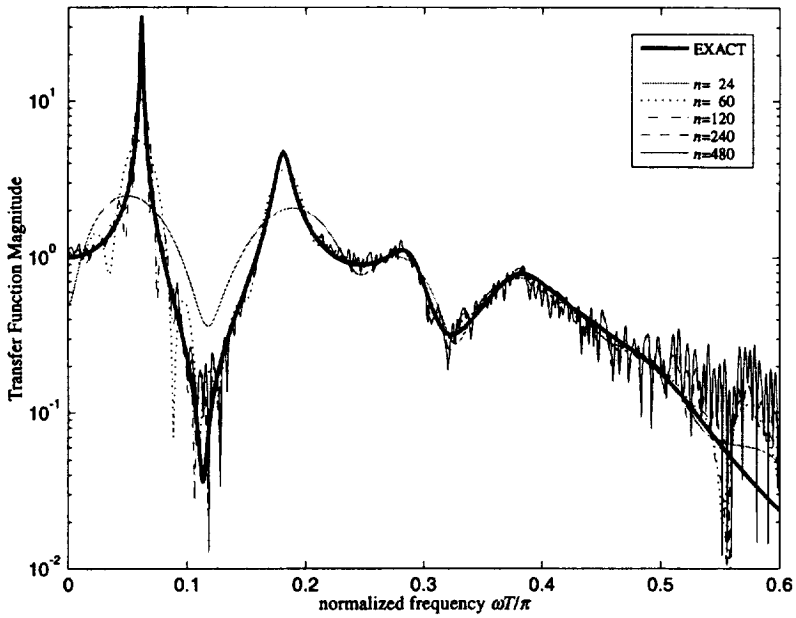


Figure 24: Magnitude of transfer function #3 of the exact system and the nonlinear identified models.

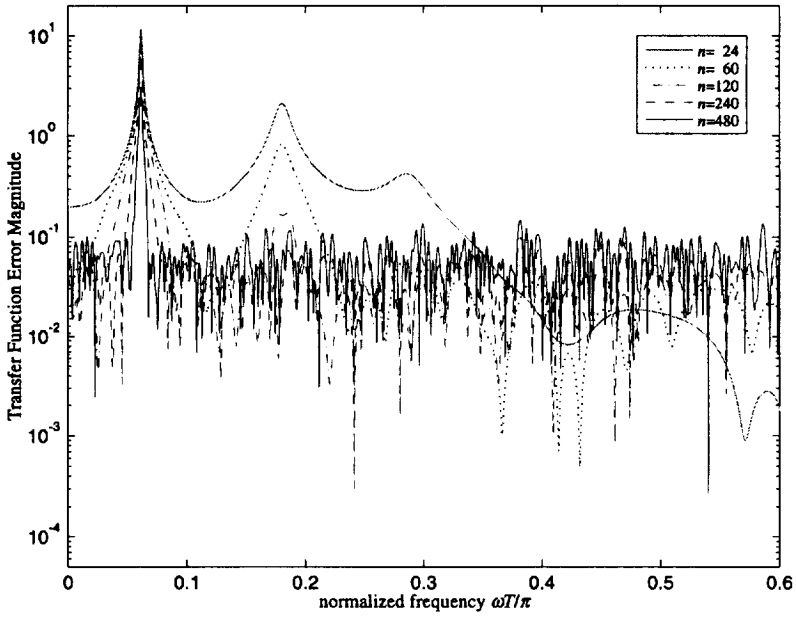


Figure 25: Magnitude of transfer function #1 error of the nonlinear identified models.

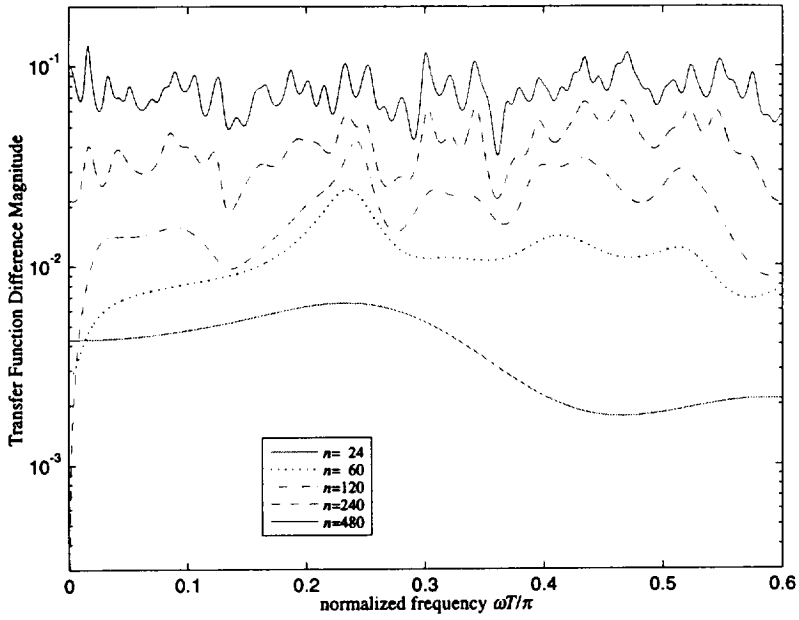


Figure 26: Magnitude of difference between linear and nonlinear identified transfer functions for output #3.

There are a number of practical concerns about this type of identification. First, the nonlinear algorithm requires the solution of an $n \times n$ eigenvalue problem. For large systems, this can be very computationally intensive and may require special care for accurate results. Second, the resulting models generally require a *significant* reduction in order. Identified models are typically 20-200 times the order of the "true" system in order to get accurate results. While order reduction is not difficult to do, it carries its own set of problems, especially for large systems. Third, a modest amount of expertise is required to properly use the H_∞ identification; for example, it has been seen (Friedman and Khargonekar, 1995a) that the choice of windowing function can be critical to the algorithm's performance. Finally, the method requires pulse response data as opposed to directly-measured input and output data; this is not a difficulty for baseline structural testing, especially since high-speed implementation of inverse discrete Fourier transform is often available, but it limits the usefulness of the method in on-line situations where speed is critical.

The conclusion, then, is that while the H_∞ identification methods lend themselves to use in some applications, on-line monitoring is not one of them for the methods studied herein.

4.2 Eigensystem Realization Algorithm

The Eigensystem Realization Algorithm (ERA) (Juang and Pappa, 1985) uses the system Markov parameters (pulse response) to compute the eigenvalues and eigenvectors of a system, which can then be used to find natural frequencies, damping ratios, and modeshapes. The system is assumed to be discrete-time, linear, and time-invariant of the form

$$\begin{aligned} \mathbf{q}(k+1) &= \mathbf{A}\mathbf{q}(k) + \mathbf{B}\mathbf{u}(k) \\ \mathbf{y}(k) &= \mathbf{C}\mathbf{q}(k) \end{aligned} \tag{78}$$

with n_u inputs and n_y outputs. The pulse response (Markov parameters), given by $\mathbf{Y}(k) = \mathbf{C}\mathbf{A}^{k-1}\mathbf{B}$, is measured, perhaps directly in the time-domain by introducing impulses into system inputs, or indirectly from the inverse discrete Fourier transform of a transfer function matrix. A generalized Hankel matrix of the Markov parameters is formed

$$\mathbf{H}(k) = \begin{bmatrix} \mathbf{Y}(k) & \mathbf{Y}(k+1) & \cdots & \mathbf{Y}(k+s) \\ \mathbf{Y}(k+1) & \mathbf{Y}(k+2) & \cdots & \mathbf{Y}(k+s+1) \\ \vdots & \vdots & & \vdots \\ \mathbf{Y}(k+r) & \mathbf{Y}(k+r+1) & \cdots & \mathbf{Y}(k+r+s) \end{bmatrix} \tag{79}$$

where r and s are arbitrary integers, but should both be at least twice the assumed order of the system for best results.

The generalized Hankel matrix for $k = 0$ is decomposed via a singular value decomposition: $\mathbf{H}(0) = \mathbf{P}\mathbf{D}\mathbf{Q}^T$. In the absence of noise, the order of the system is immediately apparent because the first n singular values are non-zero while the rest are identically zero. With noise, one must judge where the cut-off is between real and noise-induced non-zero singular values — this threshold is dependent on the estimated measurement errors and computer precision. The decomposed matrices, $\mathbf{P}$, $\mathbf{D}$, and $\mathbf{Q}$, are truncated in order to ignore the zero (or nearly zero) singular values; the first n rows and n columns of $\mathbf{D}$ are retained, as are the first n columns of both $\mathbf{P}$ and $\mathbf{Q}$.

Then, the minimum-order realization is

$$\begin{aligned} \hat{\mathbf{A}} &= \mathbf{D}^{-1/2}\mathbf{P}^T\mathbf{H}(k)\mathbf{Q}\mathbf{D}^{-1/2} \\ \hat{\mathbf{B}} &= \mathbf{D}^{1/2}\mathbf{Q}^T[\mathbf{I}_{n_u \times n_u} \ \mathbf{0}_{n_u \times sn_u}]^T \\ \hat{\mathbf{C}} &= [\mathbf{I}_{n_y \times n_y} \ \mathbf{0}_{n_y \times rn_y}]\mathbf{P}\mathbf{D}^{1/2} \end{aligned} \tag{80}$$

and the eigenvalues $\hat{z}_r$ and eigenvectors $\hat{\boldsymbol{\psi}}_r$ can be found from $\hat{\mathbf{A}}\hat{\boldsymbol{\psi}}_r = \hat{z}_r\hat{\boldsymbol{\psi}}_r$. The continuous-time poles are then given by $\hat{s}_r = (\ln\hat{z}_r \pm 2m\pi\sqrt{-1})/kT$, where T is the sampling period and m is an integer; modeshapes are given by $\hat{\mathbf{C}}\hat{\boldsymbol{\Psi}}$. k is generally chosen to be 1 for simplicity.

The modal amplitude coherence γ_r, a measure of whether a mode is judged to be true or noise-induced (γ_r is always in [0,1]; nearer zero signifies a noise-induced mode, nearer one signifies a true mode), can be computed from

$$\gamma_r^2 = \frac{|\bar{\mathbf{g}}_r^* \mathbf{g}_r|^2}{|\bar{\mathbf{g}}_r^* \bar{\mathbf{g}}_r||\mathbf{g}_r^* \mathbf{g}_r|} \tag{81}$$

where $(\ )^*$ denotes complex conjugate transpose and the column vectors $\mathbf{g}_1$, $\mathbf{b}_1$, and $\bar{\mathbf{g}}_r$ are defined by

$$\begin{aligned}
[\mathbf{g}_1 \ \mathbf{g}_2 \ \cdots \ \mathbf{g}_n]^* &= \hat{\Psi}^{-1}\mathbf{D}^{1/2}\mathbf{Q}^T \\
[\mathbf{b}_1 \ \mathbf{b}_2 \ \cdots \ \mathbf{b}_n]^* &= \hat{\Psi}^{-1}\mathbf{D}^{1/2}\mathbf{Q}^T[\mathbf{I}_{n_u \times n_u} \ \mathbf{0}_{n_u \times sn_u}]^T \\
\bar{\mathbf{g}}_r^* &= \lfloor \mathbf{b}_r^* \ \ \mathbf{b}_r^*\hat{z}_r \ \ \mathbf{b}_r^*\hat{z}_r^2 \ \ \cdots \ \ \mathbf{b}_r^*\hat{z}_r^s \rfloor
\end{aligned} \tag{82}$$

The ERA method, like the H_∞-based identification, certainly is useful for many applications. It is noted for numerical robustness due to the use of the singular value decomposition, though care is required that sufficient data is used to form the Hankel matrix for correct estimates of the system characteristics. The use of ERA and its faster variants (*e.g.*, Peterson, 1995) for on-line monitoring, however, is limited since it can be somewhat computationally intensive and may not be sufficiently fast to meet on-line requirements.

4.3 Parametric Time-Domain Methods

The methods available to do parametric identification in the time domain are probably some of the most-widely classes of system identification. Certainly, one of the reasons for this is their ease of use in general; and perhaps just as importantly, access to these routines in software packages, MATLAB® for example, has facilitated use by practitioners. Furthermore, time domain methods can, in general, better distinguish between modes whose frequencies are closely spaced than can frequency domain methods (Inman, 1989).

Most of these methods (Ljung, 1987) choose a set of parameters by using the *prediction error*, the difference between what the model predicts the output should be at some instant of time and what is actually measured at that time; *i.e.*,

$$\boldsymbol{\varepsilon}(k, \boldsymbol{\theta}) = \mathbf{y}(k) - \hat{\mathbf{y}}(k, \boldsymbol{\theta}) \tag{83}$$

where $\boldsymbol{\theta}$ is the parameter vector, $\mathbf{y}(k)$ is the measured output, and $\hat{\mathbf{y}}(k, \boldsymbol{\theta})$ is the output predicted by the model. A cost function based on the prediction error is then minimized to find the best choice of parameters $\boldsymbol{\theta}_*(k)$ based on all data up to, and including, time k

$$\boldsymbol{\theta}_*(k) = \arg\min_{\boldsymbol{\theta}} \sum_{j=1}^{k} l(j, k, \mathbf{L}(z)\boldsymbol{\varepsilon}(k, \boldsymbol{\theta}), \boldsymbol{\theta}) \tag{84}$$

where $\mathbf{L}(z)$ is a matrix of stable linear filters and $l(k, k_{\text{final}}, \boldsymbol{\varepsilon}_{\text{filtered}}, \boldsymbol{\theta})$ is a scalar-valued function (typically positive). $\mathbf{L}$ is essentially a frequency weighting of the error; for example, it can be used to prefilter the error to remove high-frequency disturbances that are not essential to the modelling or low-frequency drift or bias effects. If the predictor is linear, then using the prefilter is identical to prefiltering the input and output with the same linear filter before using the predictor. Thus, it will be assumed in the following that no prefilter is used; *i.e.*, $\mathbf{L}(z) \equiv \mathbf{I}$.

Now, the choice of the function l determines the solution. Several choices of these functions result in the special cases that follow.

The predictor can, of course, be any function of past inputs and outputs and the parameter vector $\boldsymbol{\theta}$. A convenient predictor, however, is a linear regression model

$$\hat{\mathbf{y}}(k, \boldsymbol{\theta}) = \boldsymbol{\Phi}^{\mathrm{T}}(k)\boldsymbol{\theta} + \boldsymbol{\mu}(k) \tag{85}$$

where $\boldsymbol{\Phi}(k)$ is the *regression matrix* and $\boldsymbol{\mu}(k)$ is a known function of time (and perhaps past input and output data); $\boldsymbol{\mu}(k)$ will be assumed zero for simplicity, but its exclusion leads to no loss of generality. The predictor and the prediction error, then, are

$$\begin{aligned} \hat{\mathbf{y}}(k, \boldsymbol{\theta}) &= \boldsymbol{\Phi}^{\mathrm{T}}(k)\boldsymbol{\theta} \\ \boldsymbol{\varepsilon}(k, \boldsymbol{\theta}) &= \mathbf{y}(k) - \boldsymbol{\Phi}^{\mathrm{T}}(k)\boldsymbol{\theta} \end{aligned} \tag{86}$$

The form of the regression is dependent on the assumed model of the system. Two forms that are very convenient for linear, (locally) time-invariant systems are the Auto Regressive with eXogenous inputs (ARX) and Auto Regressive Moving Average with eXogenous inputs (ARMAX) models. Both assume a linear difference equation between system inputs and outputs. The former assumes that the noise is simply an uncorrelated white noise on the sensor outputs

$$\mathbf{y}(k) + \mathbf{A}_1\mathbf{y}(k-1) + \ldots + \mathbf{A}_{n_a}\mathbf{y}(k-n_a) = \mathbf{B}_1\mathbf{u}(k-1) + \ldots + \mathbf{B}_{n_b}\mathbf{u}(k-n_b) + \mathbf{e}(k) \tag{87}$$

while ARMAX assumes a more complex noise mode, replacing the $\mathbf{e}(k)$ term with

$$\mathbf{e}(k) + \mathbf{C}_1\mathbf{e}(k-1) + \ldots + \mathbf{C}_{n_c}\mathbf{e}(k-n_c) \tag{88}$$

which allows for the sensor noise to be filtered and for white process noise. Other forms are sometimes useful, such as the Box-Jenkins and Output-Error models (Ljung, 1987).

The regression matrices for the ARX and ARMAX models are, respectively,

$$\boldsymbol{\Phi}_{\mathrm{ARX}}(k) = [-\mathbf{y}(k-1) \;\; -\mathbf{y}(k-2) \;\; \ldots \;\; -\mathbf{y}(k-n_a) \;\; \mathbf{u}(k-1) \;\; \mathbf{u}(k-2) \;\; \ldots \;\; \mathbf{u}(k-n_b)]^{\mathrm{T}} \tag{89}$$

$$\boldsymbol{\Phi}_{\mathrm{ARMAX}}(k) = [\boldsymbol{\Phi}^{\mathrm{T}}_{\mathrm{ARX}}(k) \;\; \boldsymbol{\varepsilon}(k-1, \boldsymbol{\theta}) \;\; \boldsymbol{\varepsilon}(k-2, \boldsymbol{\theta}) \;\; \ldots \;\; \boldsymbol{\varepsilon}(k-n_c, \boldsymbol{\theta})]^{\mathrm{T}} \tag{90}$$

4.3.1 Time-Domain Least-Squares Methods

The least-squares methods construct the cost function by letting l be

$$l(k, k_{\mathrm{final}}, \boldsymbol{\varepsilon}_{\mathrm{filtered}}, \boldsymbol{\theta}) = \frac{1}{2k_{\mathrm{final}}}\boldsymbol{\varepsilon}^{\mathrm{T}}(k, \boldsymbol{\theta})\mathbf{W}^{-1}(k, k_{\mathrm{final}}, \boldsymbol{\theta})\boldsymbol{\varepsilon}(k, \boldsymbol{\theta}) \tag{91}$$

where $\mathbf{W}(k, k_{\mathrm{final}}, \boldsymbol{\theta})$ is a positive, symmetric, semidefinite weighting matrix that may depend on time and (less frequently) the parameterization. This leads to a relatively simple solution for the parameters $\boldsymbol{\theta}_*$.

$$\boldsymbol{\theta}_*^{\mathrm{LS}}(k) = \left[\frac{1}{k}\sum_{j=1}^{k}\boldsymbol{\Phi}(j)\mathbf{W}^{-1}(j,k,\boldsymbol{\theta})\boldsymbol{\Phi}^{\mathrm{T}}(j)\right]^{-1}\frac{1}{k}\sum_{j=1}^{k}\boldsymbol{\Phi}(j)\mathbf{W}^{-1}(j,k,\boldsymbol{\theta})\mathbf{y}(j) \tag{92}$$

This is what is sometimes termed *weighted least-squares identification.* Of course $\mathbf{W}(k, k_{\text{final}}, \boldsymbol{\theta})$ may be the identity matrix in which case there is no weighting.

For multi-output systems, a slightly different parameterization may be possible, where the prediction regression is

$$\hat{\mathbf{y}}(k, \boldsymbol{\Theta}) = \boldsymbol{\Theta}^{\mathrm{T}}\boldsymbol{\phi}(k) \tag{93}$$

where $\boldsymbol{\Theta}$ is a $p \times n_y$ matrix. The weighted least-squares solution to $\boldsymbol{\Theta}$ is then

$$\boldsymbol{\Theta}_*^{\mathrm{LS}}(k) = \left[\frac{1}{k}\sum_{j=1}^{k}\boldsymbol{\phi}(j)\mathbf{W}^{-1}(j,k,\boldsymbol{\theta})\boldsymbol{\phi}^{\mathrm{T}}(j)\right]^{-1}\frac{1}{k}\sum_{j=1}^{k}\boldsymbol{\phi}(j)\mathbf{W}^{-1}(j,k,\boldsymbol{\theta})\mathbf{y}^{\mathrm{T}}(j) \tag{94}$$

which requires the inverse of a $p \times p$ matrix (the regression vector $\boldsymbol{\phi}$ is $p \times 1$), much quicker than inverting the $pn_y \times pn_y$ matrix in (92) (where the regression matrix $\boldsymbol{\Phi}$ is $pn_y \times n_y$).

4.3.2 Recursive Least-Squares Methods

A general recursive identification algorithm requires that it be possible to cast the parameter estimation in the form

$$\begin{aligned}\mathbf{X}(k) &= \mathbf{H}(k, \mathbf{X}(k-1), \mathbf{y}(k), \mathbf{u}(k))\\ \hat{\boldsymbol{\theta}}(k) &= \mathbf{h}(\mathbf{X}(k))\end{aligned} \tag{95}$$

where $\mathbf{H}$ and $\mathbf{h}$ are known functions that can be computed in a known amount of time (typically, less than one sampling period) and $\mathbf{X}$ is a matrix of fixed size that represents some accumulated information or knowledge. Since the amount of information in the newest measurements is small compared to the accumulated information, a more typical form is

$$\begin{aligned}\hat{\boldsymbol{\theta}}(k) &= \hat{\boldsymbol{\theta}}(k-1) + \gamma(k)\underset{\sim}{\mathbf{h}}(\mathbf{X}(k-1), \mathbf{y}(k), \mathbf{u}(k))\\ \mathbf{X}(k) &= \mathbf{X}(k-1) + \mu(k)\underset{\sim}{\mathbf{H}}(\mathbf{X}(k-1), \mathbf{y}(k), \mathbf{u}(k))\end{aligned} \tag{96}$$

where γ and μ are small numbers that reflect the information content of the latest measurements. For many methods of interest, $\underset{\sim}{\mathbf{h}}$ has a simple form such that

$$\hat{\boldsymbol{\theta}}(k) = \hat{\boldsymbol{\theta}}(k-1) + \mathbf{K}(k)\left[\mathbf{y}(k) - \hat{\mathbf{y}}(k, \boldsymbol{\theta})\right] \tag{97}$$

A weighted least-squares algorithm, where the weighting $\mathbf{W}(k, k_{\text{final}}, \boldsymbol{\theta})$ is of the form

$$\begin{aligned} \mathbf{W}^{-1}(k\quad, k_{\text{final}}, \boldsymbol{\theta}) &= \lambda(k)\mathbf{W}^{-1}(k-1, k_{\text{final}}, \boldsymbol{\theta}) \\ \mathbf{W}^{-1}(k_{\text{final}}, k_{\text{final}}, \boldsymbol{\theta}) &= \mathbf{I} \end{aligned} \tag{98}$$

or, equivalently,

$$\mathbf{W}^{-1}(k, k_{\text{final}}, \boldsymbol{\theta}) = \mathbf{I} \prod_{j=k+1}^{k_{\text{final}}} \frac{1}{\lambda(j)}, \tag{99}$$

can be cast in the recursive relationship

$$\begin{aligned} \hat{\boldsymbol{\theta}}(k) &= \hat{\boldsymbol{\theta}}(k-1) + \mathbf{R}^{-1}(k)\boldsymbol{\Phi}(k)\,[\mathbf{y}(k) - \boldsymbol{\Phi}^{\mathrm{T}}(k)\hat{\boldsymbol{\theta}}(k-1)] \\ \mathbf{R}(k) &= \lambda(k)\mathbf{R}(k-1) + \boldsymbol{\Phi}(k)\boldsymbol{\Phi}^{\mathrm{T}}(k) \end{aligned} \tag{100}$$

Another convenient form is found by letting $\mathbf{P}(k) = \mathbf{R}^{-1}(k)$, applying the matrix inversion lemma to the equation for $\mathbf{R}(k)$, and simplifying

$$\begin{aligned} \hat{\boldsymbol{\theta}}(k) &= \hat{\boldsymbol{\theta}}(k-1) + \mathbf{P}(k-1)\mathbf{L}(k)\,[\mathbf{y}(k) - \boldsymbol{\Phi}^{\mathrm{T}}(k)\hat{\boldsymbol{\theta}}(k-1)] \\ \mathbf{L}(k) &= \boldsymbol{\Phi}(k)\,[\lambda(k)\mathbf{I} + \boldsymbol{\Phi}^{\mathrm{T}}(k)\mathbf{P}(k-1)\boldsymbol{\Phi}(k)]^{-1} \\ \mathbf{P}(k) &= \frac{1}{\lambda(k)}\mathbf{P}(k-1)\,\{\mathbf{I} - \mathbf{L}(k)\boldsymbol{\Phi}^{\mathrm{T}}(k)\mathbf{P}(k-1)\} \end{aligned} \tag{101}$$

This form is especially convenient for single output systems since the matrix to be inverted in (101) is a scalar, whereas in (100) it is $p \times p$.

This variant is often called the *forgetting factor* method since it facilitates weighting past information exponentially less as time goes on, thus allowing identification of slowly time-varying systems. A constant forgetting factor $\lambda = \lambda(k)$ is often used, making the weight

$$\mathbf{W}^{-1}(k, k_{\text{final}}, \boldsymbol{\theta}) = \lambda^{k-k_{\text{final}}}\,\mathbf{I} \tag{102}$$

Several other recursive least-squares variants are available. One, which assumes that the change in the true parameters $\boldsymbol{\theta}_0(k)$ follows a random walk, is given by

$$\boldsymbol{\theta}_0(k) = \boldsymbol{\theta}_0(k-1) + \text{white noise} \tag{103}$$

A Kalman filter formulation can then be used to minimize the error in the estimated parameters. Two additional variants are the *unnormalized* and *normalized gradient* approaches, which set $\mathbf{K}(k)$ in (97) to

$$\mathbf{K}(k) = \begin{cases} \Upsilon\boldsymbol{\psi}(k), & \text{unnormalized gradient} \\ \Upsilon\boldsymbol{\psi}(k)|\boldsymbol{\psi}(k)|^{-2}, & \text{normalized gradient} \end{cases} \tag{104}$$

where $\boldsymbol{\psi}(t)$ is an estimate of the gradient of $\hat{\mathbf{y}}(t, \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$.

One significant concern is the computational complexity of the recursive algorithms, which must be low enough to allow on-line processing of the data. The complexity of the recursive least-squares methods goes as p^2 where the parameter vector $\boldsymbol{\theta}$ is $p \times 1$. For large systems, this may seem prohibitive, but several faster modified recursive least-squares algorithms exist (Ljung and Söderström, 1983, especially Appendix 6; Ljung, 1987) whose complexity is closer to $O(p)$, for example *fast transversal filters* (Cioffi & Kailath, 1984) and *least-squares lattice filters* (Griffiths, 1977; Makhoul, 1977; Lee *et al.*, 1981).

It should be noted that MATLAB® does include a number of recursive least-square routines, including all four variants mentioned above. Both ARX and ARMAX system models are included, along with Box-Jenkins, Output-Error, and a general Prediction-Error model. Its implementation, however, is limited to single-input, single-output (SISO) systems for many of the routines, and to multi-input, single-output (MISO) for others. None of the recursive least-squares algorithms will handle the general multi-input, multi-output (MIMO) systems; perhaps MIMO systems were not included due to the matrix inversion in (100) or (101) if the output is not scalar. One could argue that a model for each output of a system could be estimated and then combined. Generally, however, the dynamics for each output are linked. With all outputs handled at the same time, one would expect better results due to the additional information available to the estimator.

4.3.3 Instrumental Variable Methods

An alternate formulation can be based on requiring that the noise be uncorrelated with past samples. This may be formulated by requiring

$$\frac{1}{k}\sum_{j=1}^{k} \mathbf{V}(j)\,[\mathbf{y}(j) - \boldsymbol{\Phi}^{\mathrm{T}}(k)\boldsymbol{\theta}] \;=\; 0 \tag{105}$$

where $\mathbf{V}(j)$ are the *instrumental variables* that should be correlated with the regression matrix but uncorrelated with the noise

$$\begin{aligned} &E\,[\mathbf{V}(k)\boldsymbol{\Phi}^{\mathrm{T}}(k)] \text{ is nonsingular} \\ &E\,[\mathbf{V}(k)\mathbf{v}(k)] \;=\; 0 \end{aligned} \tag{106}$$

where the measured data is assumed to be given by

$$\mathbf{y}(k) \;=\; \boldsymbol{\Phi}^{\mathrm{T}}(k)\boldsymbol{\theta} + \mathbf{v}(k) \tag{107}$$

Appropriate instrumental variables for ARX and ARMAX models can be found. One method of constructing the instrumental variables with an ARX model is to first use a least-squares search and use the resulting estimation of $\mathbf{A}_j^{\mathrm{LS}}$ and $\mathbf{B}_j^{\mathrm{LS}}$ to construct the instrumental variables

$$\mathbf{V}(k) \;=\; [-\mathbf{x}(k-1)\;\; -\mathbf{x}(k-2)\;\; \ldots\;\; -\mathbf{x}(k-n_a)\;\; \mathbf{u}(k-1)\;\; \mathbf{u}(k-2)\;\; \ldots\;\; \mathbf{u}(k-n_b)]^{\mathrm{T}} \tag{108}$$

where $\mathbf{x}$ is the input filtered through the ARX model

$$[\mathbf{I} + \mathbf{A}_1^{LS} z^{-1} + \ldots + \mathbf{A}_{n_a}^{LS} z^{-n_a}]\,\mathbf{x}(k) \;=\; [\mathbf{B}_1^{LS} z^{-1} + \ldots + \mathbf{B}_{n_b}^{LS} z^{-n_b}]\,\mathbf{u}(k) \tag{109}$$

Instrumental variable methods can be done recursively. For details on this, on other choices of the instrumental variables, and a full derivation, see Ljung (1987) and the references therein.

4.4 Comparison of Various System Identification Methods

A number of the aforementioned system identification methods have been evaluated for their usefulness in on-line monitoring of structural systems. Table 3 rates these methods according to (*i*) the expertise required to use the method well, (*ii*) numerical convergence properties of the method, (*iii*) the potential for use on-line, (*iv*) where the initial guess must be, (*v*) the reliability of the results, and (*vi*) what knowledge is required *a priori*. (Part of this chart is borrowed from Shinozuka and Ghanem (1995).)

The method that best appears to meet the criteria for on-line monitoring of slowly time-varying systems is a recursive least-squares method that uses the "forgetting factor" variant (which weights past measurements exponentially less). This method is easy to use, produces reliable results, and has high on-line potential. Furthermore, it is able to handle time-varying systems well due to the exponential weight.

ID Method	*Required Expertise*	*Numerical Convergence*	*On-line Potential*	*Initial Guess*	*Reliability of Results*	*a priori knowledge*
ERA	medium	good	low	anywhere	good	pulse resp.
H_∞	substantial	usually	low	anywhere	medium	pulse resp.
Least Squares (ARX, ARMAX, etc.)	minimal	always	low	anywhere	good	I/O hist; order
Recursive Least Squares (RARX, etc.)	minimal	always	high	anywhere	medium	I/O hist; order
Recursive Least Squares (RARX, etc.) with exponential window	**minimal**	**always**	**high**	**anywhere**	**good**	**I/O hist; order**
Maximum Likelihood	substantial	sometimes	low	close	good	I/O hist; order
Recursive Instrumental variable	medium	always	high	anywhere	medium	I/O hist; order
Extended Kalman Filter	substantial	sometimes	low	close	good	*varies*
Subspace Methods (N4SID)	minimal	always	low	anywhere	good	I/O hist.

Table 3: Evaluation and comparison of some system identification methods

5.0 TWO-STAGE ADAPTIVE MONITORING

Given that available on-line computer power is constant, the goal of accurate identification of structural parameters and/or modal responses is generally in direct contradiction with attempting to update such information rapidly to track a time-varying system. A two-stage adaptive monitoring scheme can meet the sometimes conflicting needs of on-line monitoring.

5.1 BASIC DESIGN OF A TWO-STAGE ALGORITHM

Two loops characterize a two-stage algorithm. The outer loop, which is performed on-line but not necessarily in real time, is a system identification loop that updates, as often as possible, estimates of the system parameters and a Kalman filter to monitor the system. The real-time inner loop uses the most recently available Kalman filter output to monitor modal response; this modal response may be used to watch for pathological behavior of certain modes, or it may be used as the input to a simple single degree of freedom identification algorithm to update frequency and damping more rapidly than the outer loop can. Figure 27 shows a block diagram of a two-stage algorithm. Similar multi-stage algorithms (*e.g.*, Chen *et al.*, 1992) for the purpose of adaptive control have been studied.

Note that the method used to identify the system is not necessarily defined. It may be a standard algorithm, such as a recursive least-squares time-domain method, or something more esoteric.

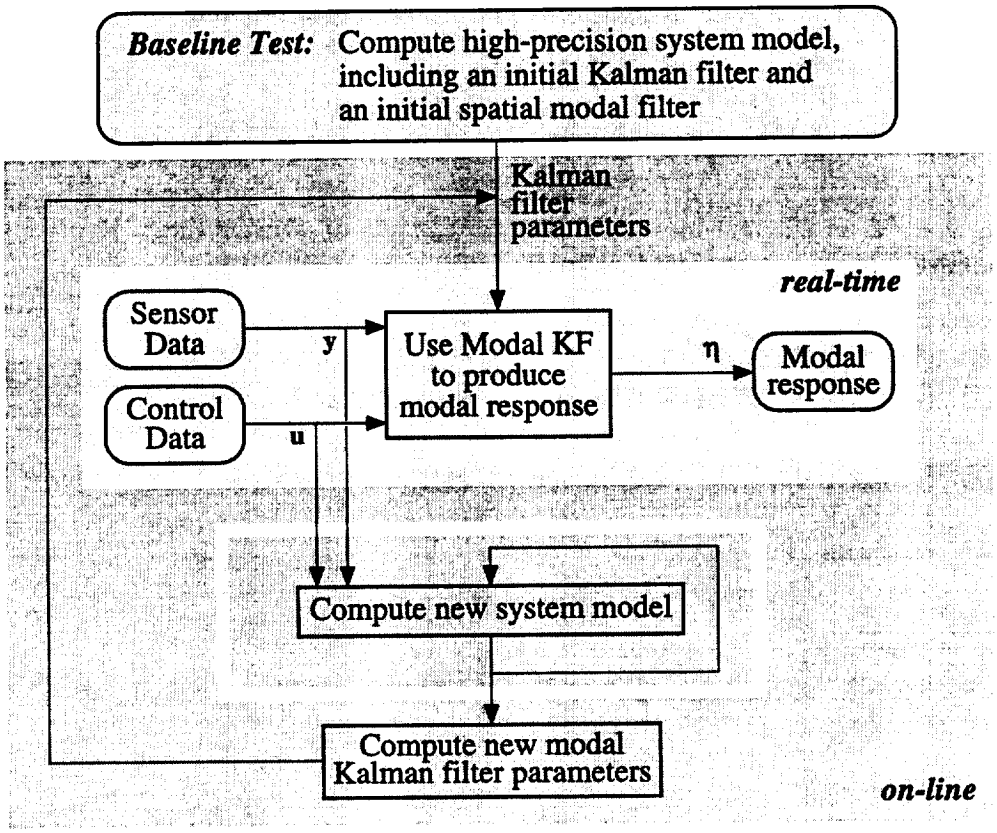


Figure 27: Block diagram of a two-stage adaptive monitoring algorithm.

In order to monitor modal response, a modal Kalman filter can be used. To do so, the identified model must be put into a state-space form. A similarity transformation is used to convert the identified system to modal state-space (block diagonal form). Assume the identified system is of order n with state-space description $(\underset{\sim}{\mathbf{A}}, \underset{\sim}{\mathbf{B}}, \underset{\sim}{\mathbf{C}}, \mathbf{D})$ in continuous- or discrete-time

$$\begin{aligned} \underset{\sim}{\mathbf{x}}(k+1) &= \underset{\sim}{\mathbf{A}}^{\mathrm{d}}\,\underset{\sim}{\mathbf{x}}(k) + \underset{\sim}{\mathbf{B}}^{\mathrm{d}}\mathbf{u}(k) \\ \mathbf{y}(k) &= \underset{\sim}{\mathbf{C}}^{\mathrm{d}}\underset{\sim}{\mathbf{x}}(k) + \mathbf{D}^{\mathrm{d}}\mathbf{u}(k) \end{aligned} \quad \text{or} \quad \begin{aligned} \dot{\underset{\sim}{\mathbf{x}}}(t) &= \underset{\sim}{\mathbf{A}}^{\mathrm{c}}\,\underset{\sim}{\mathbf{x}}(t) + \underset{\sim}{\mathbf{B}}^{\mathrm{c}}\mathbf{u}(t) \\ \mathbf{y}(t) &= \underset{\sim}{\mathbf{C}}^{\mathrm{c}}\underset{\sim}{\mathbf{x}}(t) + \mathbf{D}^{\mathrm{c}}\mathbf{u}(t) \end{aligned} \tag{110}$$

Let λ_r and $\boldsymbol{\Phi}$ be the right eigenvalues and eigenmatrix of $\underset{\sim}{\mathbf{A}}$, arranged such that

$$\begin{aligned} \boldsymbol{\Phi} &= [\phi_1\ \phi_1^*|\ldots|\phi_p\ \phi_p^*\ |\ \psi_1\ldots\psi_q], \qquad n = 2p+q \\ \boldsymbol{\lambda} &= [\lambda_1\ \lambda_1^*|\ldots|\lambda_p\ \lambda_p^*|\ \mu_1\ldots\mu_q]^{\mathrm{T}} \end{aligned} \tag{111}$$

Then the similarity transformation to convert to modal state-space is

$$\mathbf{T} = \boldsymbol{\Phi}\begin{bmatrix} -a_1^* & 1 & & & & \mathbf{0} \\ -a_1 & 1 & & & & \\ & \ddots & & & & \\ & & -a_p^* & 1 & & \\ & & -a_p & 1 & & \\ \mathbf{0} & & & & & \mathbf{I}_{q\times q} \end{bmatrix} \qquad a_i = \begin{cases} \lambda_i, \text{ continuous} \\ \frac{1}{T}\ln\lambda_i, \text{ discrete} \end{cases} \tag{112}$$

where the a_i depend on whether the model is in continuous or discrete time. Replacing $\underset{\sim}{\mathbf{x}}$ with $\mathbf{Tx}$ and premultiplying the state equations by $\mathbf{T}^{-1}$ will give

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}^{\mathrm{d}}\,\mathbf{x}(k) + \mathbf{B}^{\mathrm{d}}\mathbf{u}(k) \\ \mathbf{y}(k) &= \mathbf{C}^{\mathrm{d}}\mathbf{x}(k) + \mathbf{D}^{\mathrm{d}}\mathbf{u}(k) \end{aligned} \quad \text{or} \quad \begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}^{\mathrm{c}}\,\mathbf{x}(t) + \mathbf{B}^{\mathrm{c}}\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}^{\mathrm{c}}\mathbf{x}(t) + \mathbf{D}^{\mathrm{c}}\mathbf{u}(t) \end{aligned} \tag{113}$$

which is a system whose states are modal displacements and velocities

$$\mathbf{A} = \begin{bmatrix} \begin{bmatrix} \mathbf{A}_1 \end{bmatrix} & & & & \mathbf{0} \\ & \ddots & & & \\ & & \begin{bmatrix} \mathbf{A}_p \end{bmatrix} & & \\ & & & \mu_1 & \\ \mathbf{0} & & & & \ddots \\ & & & & & \mu_q \end{bmatrix} \qquad \mathbf{x} = \begin{Bmatrix} \eta_1 \\ \dot{\eta}_1 \\ \hline \vdots \\ \hline \eta_p \\ \dot{\eta}_p \\ \hline \alpha_1 \\ \vdots \\ \alpha_q \end{Bmatrix} \tag{114}$$

$$\mathbf{B} = \mathbf{T}^{-1}\underset{\sim}{\mathbf{B}} \qquad \mathbf{C} = \underset{\sim}{\mathbf{C}}\mathbf{T} \tag{115}$$

where the p 2×2 blocks $\mathbf{A}_r$ in $\mathbf{A}$ are functions of modal frequency ω_r and damping ratio ζ_r

$$\mathbf{A}_r^c = \begin{bmatrix} 0 & 1 \\ -\omega_r^2 & -2\zeta_r\omega_r \end{bmatrix} \quad \text{or} \quad \mathbf{A}_r^d = e^{\mathbf{A}_r^c T} = e^{-\zeta_r\omega_r T}\left(\begin{bmatrix} \zeta_r\omega_r & 1 \\ -\omega_r^2 & -\zeta_r\omega_r \end{bmatrix} \frac{\sin\omega_{d_r} T}{\omega_{d_r}} + \mathbf{I}\cos\omega_{d_r} T \right) \tag{116}$$

where $\omega_{d_r} = \omega_r\sqrt{1-\zeta_r^2}$. The modal state-space description $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ can then be used to easily compute natural frequencies and damping ratios. It can further be used to formulate a time-varying Kalman filter to efficiently estimate modal responses.

A time-varying Kalman filter (Chui and Chen, 1987) for a system with discrete-time state-space description

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}(k)\mathbf{x}(k) + \mathbf{B}(k)\mathbf{u}(k) + \mathbf{\Gamma}(k)\boldsymbol{\xi}(k) \\ \mathbf{y}(k) &= \mathbf{C}(k)\mathbf{x}(k) + \mathbf{D}(k)\mathbf{u}(k) + \boldsymbol{\eta}(k) \end{aligned} \tag{117}$$

where zero $\boldsymbol{\xi}(k)$ and $\boldsymbol{\eta}(k)$ are zero-mean Gaussian white noise vectors with covariance

$$E\left[\left\{ \begin{array}{c} \boldsymbol{\xi}(k) \\ \hline \boldsymbol{\eta}(k) \\ \hline \mathbf{x}(0) \end{array} \right\} \left\lfloor \boldsymbol{\xi}^T(k) \mid \boldsymbol{\eta}^T(k) \right\rfloor \right] = \left[\begin{array}{c|c} \mathbf{Q}(k) & \mathbf{S}(k) \\ \hline \mathbf{S}^T(k) & \mathbf{R}(k) \\ \hline \mathbf{0} & \mathbf{0} \end{array} \right] \qquad \begin{array}{l} \mathbf{Q}(k) \geq 0 \\ \mathbf{R}(k) > 0 \\ \mathbf{S}(k) \geq 0 \end{array} \tag{118}$$

and with initial state conditions $\mathbf{x}(0)$, is given by the initial conditions

$$\hat{\mathbf{P}}(0) = E[\mathbf{x}(0)\mathbf{x}^T(0)] \qquad \hat{\mathbf{x}}(0) = E[\mathbf{x}(0)] \tag{119}$$

and the recursive filter equations (terms in gray may be eliminated if $\mathbf{S}(k) \equiv 0$)

$$\begin{aligned} \mathbf{K}(k) &= \mathbf{\Gamma}(k)\mathbf{S}(k)\mathbf{R}^{-1}(k) \\ \bar{\mathbf{P}}(k) &= [\mathbf{A}(k-1) - \mathbf{K}(k-1)\mathbf{C}(k-1)]\,\hat{\mathbf{P}}(k-1)\,[\mathbf{A}(k-1) - \mathbf{K}(k-1)\mathbf{C}(k-1)]^T \\ &\quad + \mathbf{\Gamma}(k-1)\mathbf{Q}(k-1)\mathbf{\Gamma}^T(k-1) - \mathbf{K}(k-1)\mathbf{R}(k-1)\mathbf{K}^T(k-1) \\ \mathbf{G}(k) &= \bar{\mathbf{P}}(k)\mathbf{C}^T(k)\,[\mathbf{C}(k)\bar{\mathbf{P}}(k)\mathbf{C}^T(k) + \mathbf{R}(k)]^{-1} \\ \hat{\mathbf{P}}(k) &= [\mathbf{I} - \mathbf{G}(k)\mathbf{C}(k)]\,\bar{\mathbf{P}}(k) \\ \bar{\mathbf{x}}(k) &= \mathbf{A}(k-1)\hat{\mathbf{x}}(k-1) + \mathbf{B}(k-1)\mathbf{u}(k-1) \\ &\quad + \mathbf{K}(k-1)[\mathbf{y}(k-1) - \mathbf{D}(k-1)\mathbf{u}(k-1) - \mathbf{C}(k-1)\hat{\mathbf{x}}(k-1)] \\ \hat{\mathbf{x}}(k) &= [\mathbf{I} - \mathbf{G}(k)\mathbf{C}(k)]\,\bar{\mathbf{x}}(k) - \mathbf{G}(k)\mathbf{D}(k)\mathbf{u}(k) + \mathbf{G}(k)\mathbf{y}(k) \\ \hat{\mathbf{y}}(k) &= \mathbf{C}(k)\hat{\mathbf{x}}(k) + \mathbf{D}(k)\mathbf{u}(k) \end{aligned} \tag{120}$$

5.2 EXAMPLE OF A TWO-STAGE ALGORITHM

To demonstrate the usefulness of a two-stage algorithm, it will be applied to several systems; first to a single degree of freedom system to demonstrate several important observations on the effects of various algorithmic parameters, and then to two multi-degree of freedom systems with two and six degrees of freedom, respectively. All three base systems are like the six degree of freedom system shown in Fig. 4, and have masses, spring stiffnesses, and damping coefficients

$$m_i = 1\,,\ k_i = 1\,, \text{and}\ c_i = 0.1\,, \tag{121}$$

respectively; the natural frequencies, then, are clustered around 1 rad/sec. (This is, of course, quite a bit lower than most real-world structural systems of interest here, but since it is a linear system, the time- and frequency-ranges are easily scaled.) These systems will then be modified in piecewise time-invariant and continuously time-varying manners to demonstrate the ability to track to such changing systems.

An ARX system model is assumed for several reasons. First, it is relatively simple and can be used to clarify certain issues. Second, it allows the use of a recursive algorithm RARX. Third, the identification can be implemented easily with MATLAB® via the `rarx` function. Fourth, the extension to an ARMAX model, which includes a more realistic noise model, is direct. The *forgetting factor* variant will be used since the real system will be time varying in some of the examples.

Note that all of the examples were executed with Gaussian white noise inputs (of varying magnitudes). Thus, results will vary with realizations of the noise.

5.2.1 Effect of Algorithm Parameters on a SDOF System

Before applying the recursive least-squares ARX algorithm, it is useful to see the effect of the *forgetting factor* on the identification. Figure 28 shows how much weight is given to past data at time $t = 750$ seconds. Forgetting factors λ in the range of $[0.95, 0.995]$ were examined in this study. The proper choice of the forgetting factor is dependent on the signal-to-noise ratio and how much the system is expected to change. For example, Fig. 29 shows the effect of the forgetting factor on a single-input, single-output (SISO), single degree of freedom (SDOF), time-invariant system. A higher forgetting factor is less sensitive to noise; one way to think of this is that the higher forgetting factor is using more past data in its averages, thus smoothing out the noise somewhat. On the other hand, it will be seen below that a higher forgetting factor is slower in reacting to real changes in the system for the same reason. (In what follows, if the forgetting factor is not specified, it is 0.98.)

Of course, the accuracy of any identification algorithm is dependent on the quality of the data used to do the identification. The effect of the magnitude of the sensor noise on the SISO, SDOF, time-invariant system is shown in Fig. 30, with signal-to-noise ratios of 4, 10, and 20. The frequency estimation is within a few percent of the actual even for a signal-to-noise ratio of 10; damping estimates are not as accurate (which is generally the case for most identification algorithms), being within 15-20% for a signal-to-noise ratio of 20.

One helpful step that can be taken before the identification to improve the estimator is to prefilter the input and output data through a bandpass filter. This can serve to eliminate both high

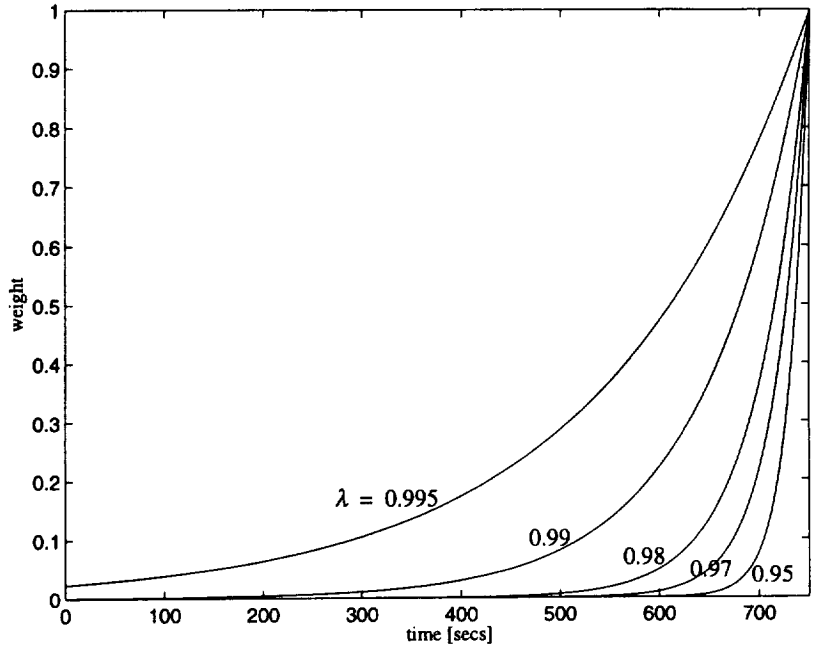


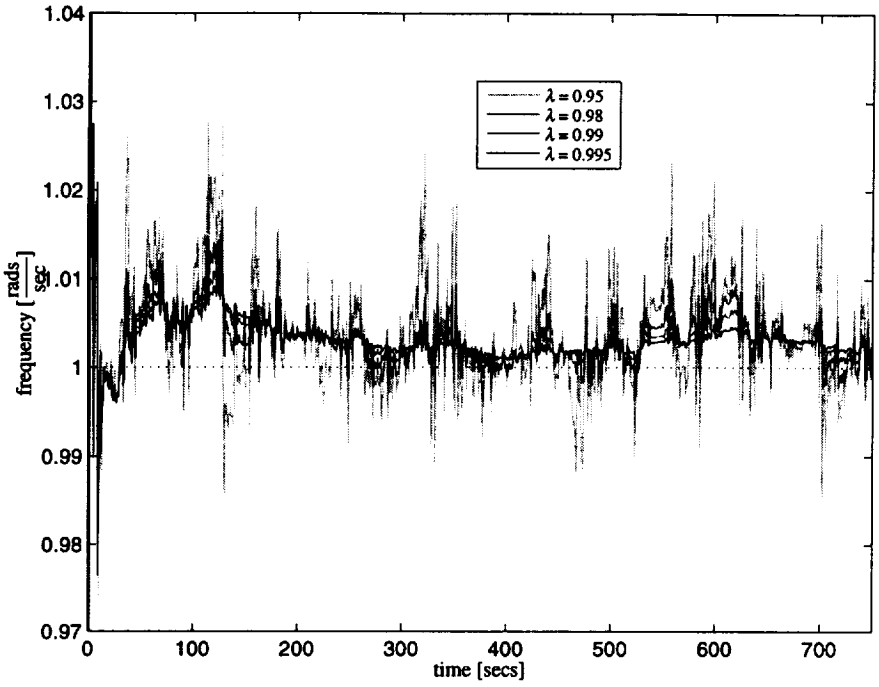
Figure 28: Weight of past samples using various *forgetting factor* values.

frequency noise that is far out of the frequency bandwidth of the structure and low frequency drift or bias that may occur due to inaccuracies in sensors and acquisition hardware. Since the data here is from simulation, the latter problem does not occur, but a lowpass filter, even a first-order filter, to eliminate the high frequency noise improves the results of the estimator. Figure 31 shows that this is true for both frequency and damping estimates with filters of several orders (all have cut-off frequency approximately 2.6 times the natural frequency of the oscillator). The results below use an eighth-order lowpass filter.

5.2.2 Tracking a Time-Varying SDOF System

To verify that a time-varying system can be monitored, frequency and damping estimates of the SISO, SDOF system are found as the system changes in several ways. Several different values of the forgetting factor are used to demonstrate how that parameter changes the response of the estimation as the system is modified. Piecewise time-invariant changes dwell for a period of time at the base system, instantaneously changes and dwells for a like duration with new system characteristics, and then changes instantaneously back to the base system; this allows an evaluation of the tracking ability of the identification to follow an instantaneous change in system configuration. Continuously time-varying changes demonstrate the ability to follow small, but continuous, changes in system characteristics.

The first modification is letting the mass instantaneously decrease by 50% for a period of time and then back to its original value. The frequency and damping estimates of this system are shown in Fig. 32. The lower forgetting factor reacts much more quickly to the change in frequency and damping, but displays significantly more noise, especially in the damping estimate. In all cases, however, the identification algorithm is able to track in on the new frequency and damping values.



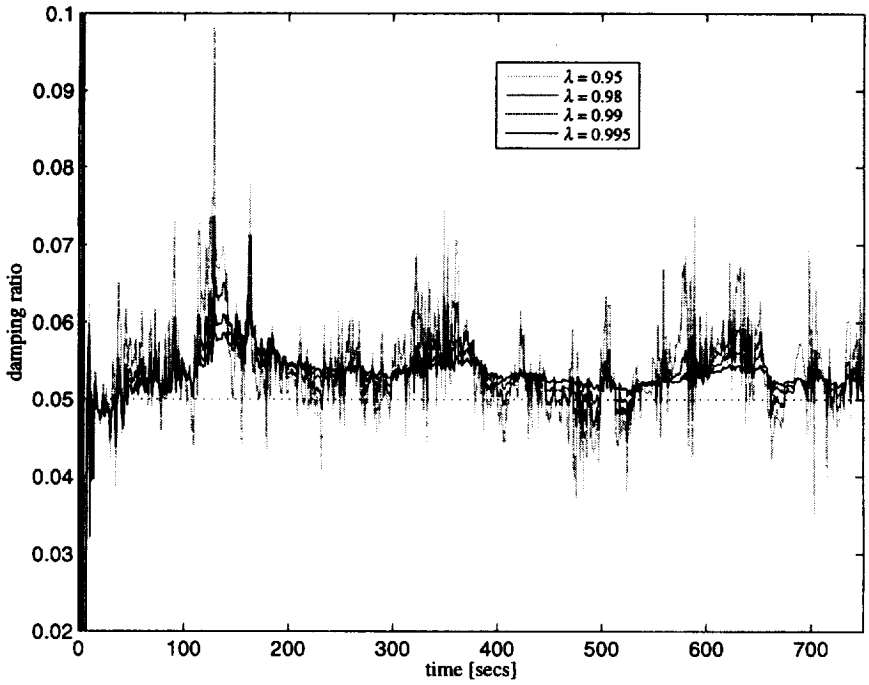


Figure 29: The effect of varying the forgetting factor on the identified frequency and damping of a SISO SDOF time-invariant system.

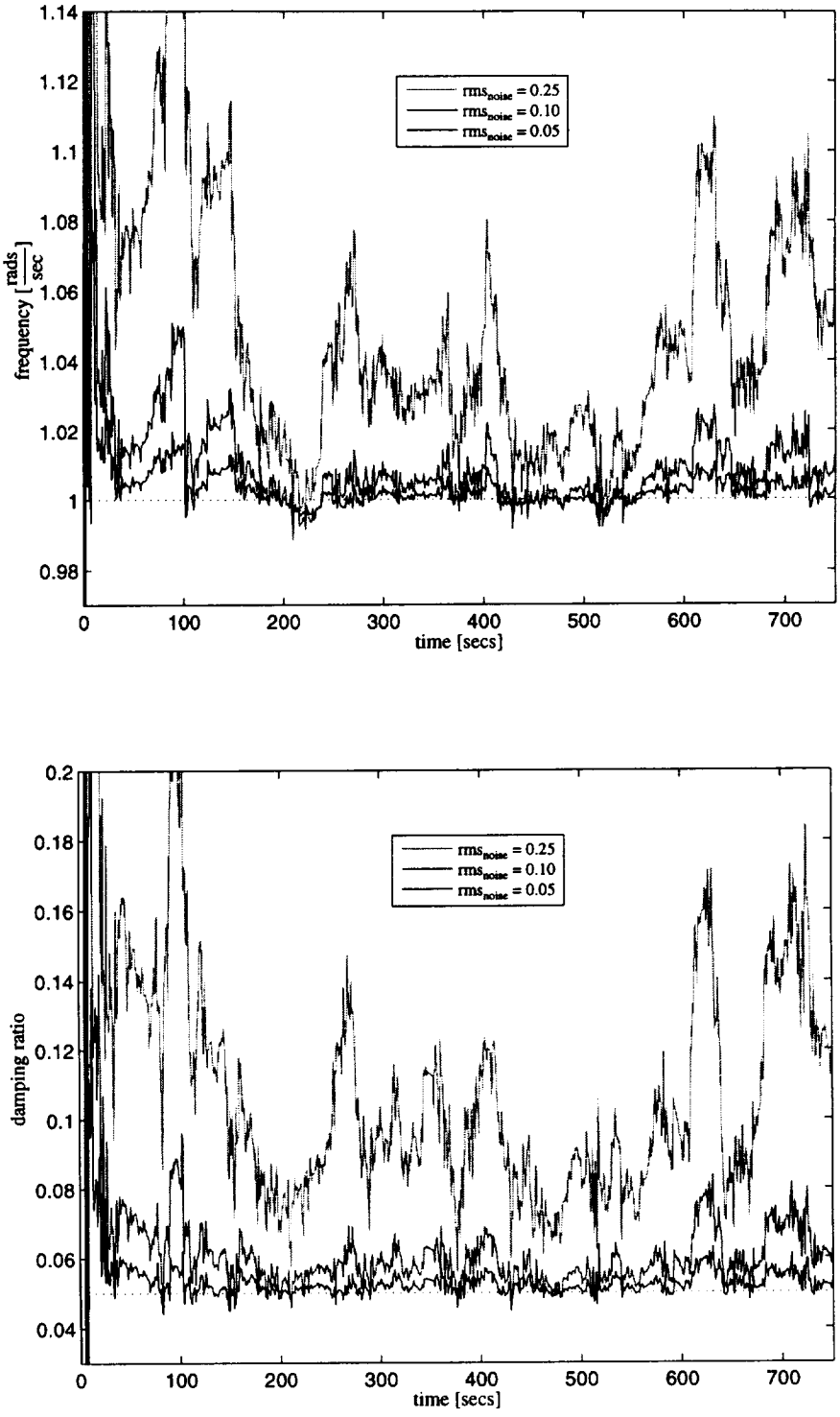


Figure 30: The effect of sensor noise magnitude on the identified frequency and damping of a SISO SDOF time-invariant system.

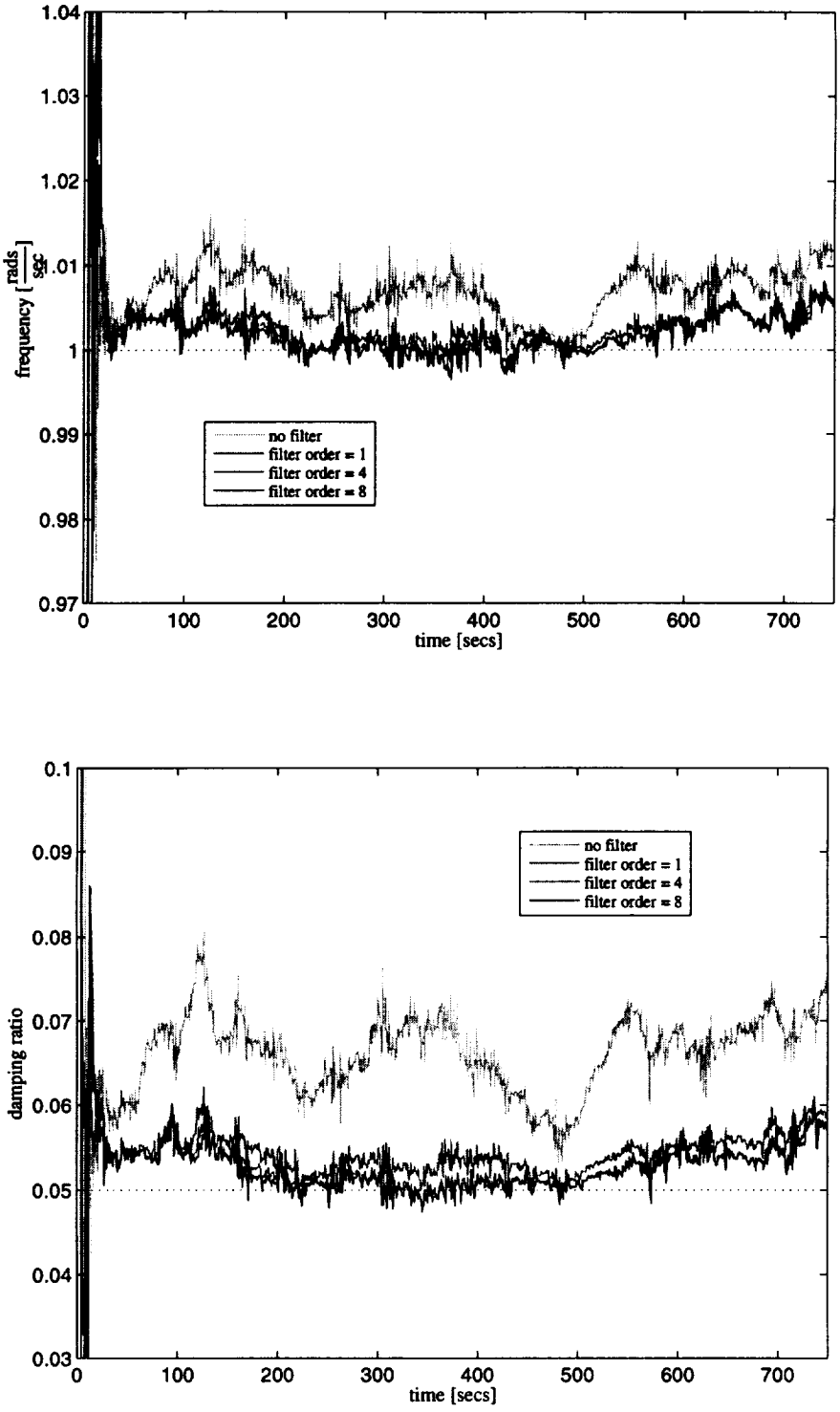


Figure 31: The effect of prefiltering with a lowpass filter of various orders on the identified frequency and damping of a SISO SDOF time-invariant system.

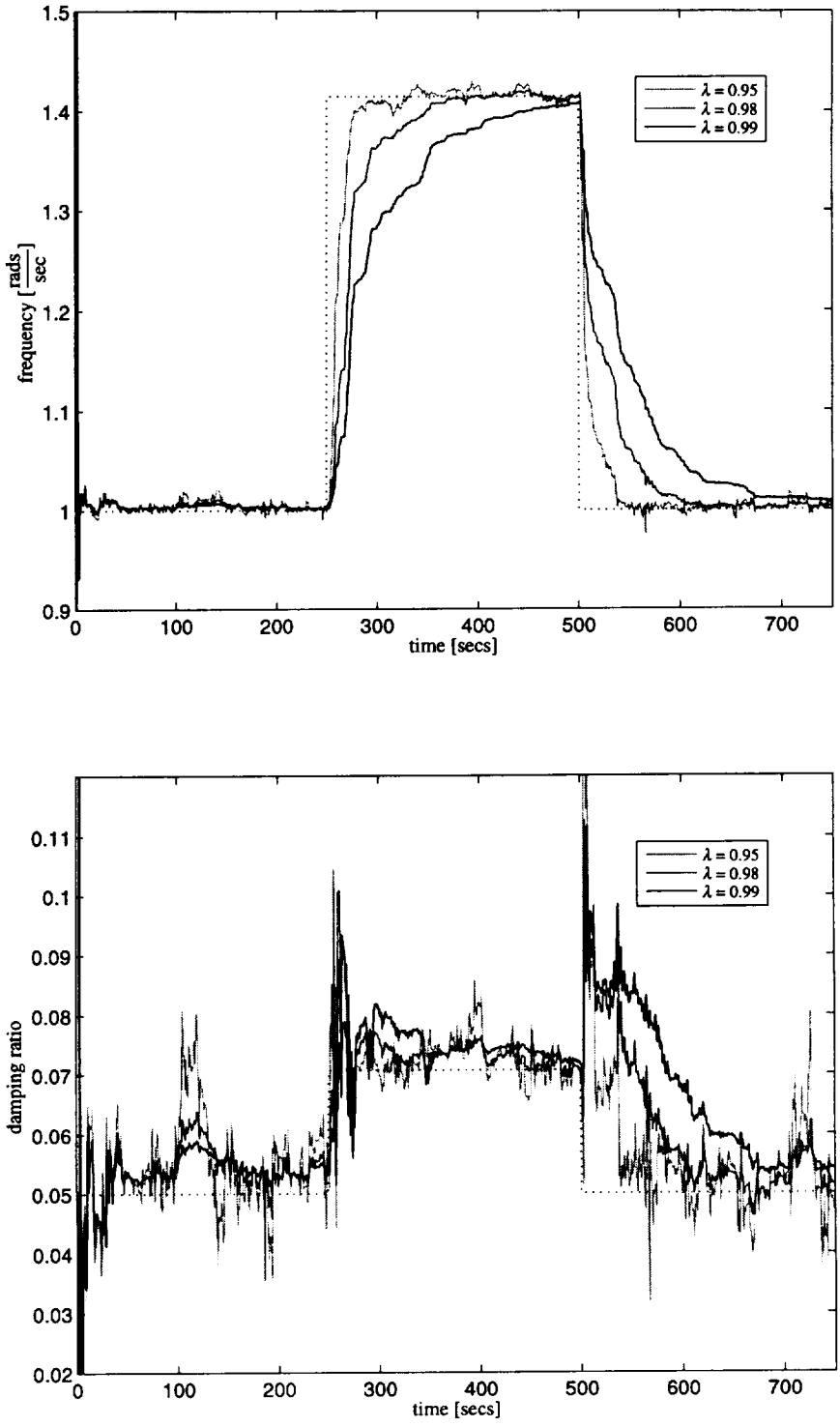


Figure 32: Tracking the frequency and damping of a piecewise time-invariant (mass decreasing) SISO SDOF system.

Figures 33 and 34 show similar systems except where the spring stiffness decreases and increases, respectively, by a factor of two, causing the frequency to change by a factor of $\sqrt{2}$. In both cases, the frequency and damping estimates settle down to something near their exact values.

Changes in damping ratios are critical for monitoring such phenomena as flutter. A decrease in damping alone gives the results in Fig. 35 where the damping is instantaneously cut in half for a period of time. The frequency estimates remain relatively constant; the damping estimate is able to track the changing damping ratio fairly well. Again, it is seen that the forgetting factor has a significant effect on the speed of tracking to the new value and a like effect on the sensitivity to noise. Figure 36 shows that a similar change, but now increasing damping by 50%, results in the same effects.

The frequency and damping estimates due to a continuously time-varying change in the mass of a SISO, SDOF system are shown in Fig. 37. The frequency tracks quite well, with some short time lag, but damping estimates demonstrate a larger time lag, especially to an increase in mass (causing a decrease in damping and frequency). Changes in damping alone, however, allows for much better damping estimates, as seen in Fig. 38. Here, the damping follows a slow sinusoidal value between 50% above and 50% below the nominal value. The damping estimate is able to track this kind of change quite well.

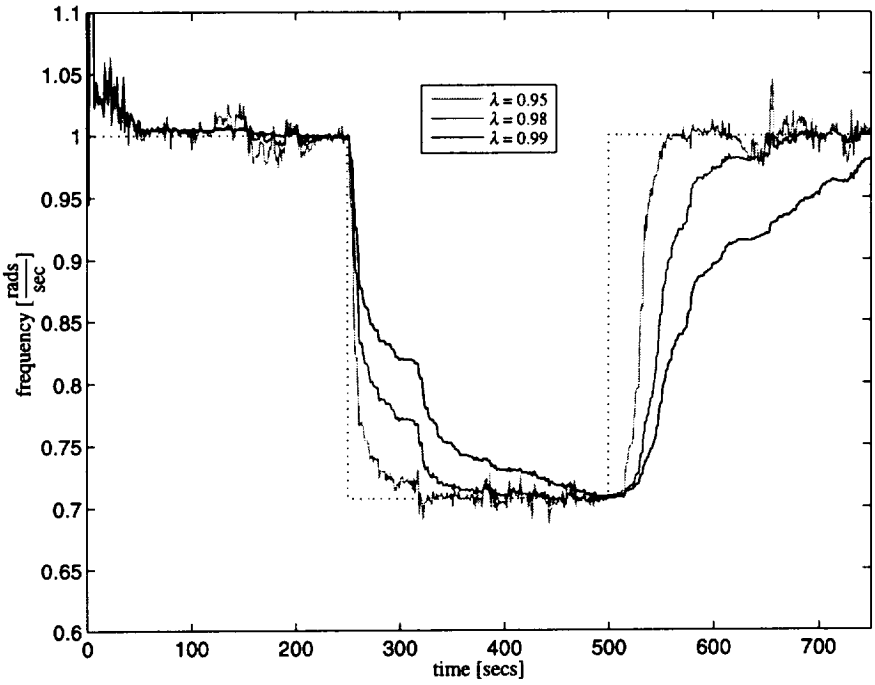
5.2.3 Tracking Two Time-Varying MDOF Systems

Most of the observations made for the single degree of freedom hold also for multi-input, multi-degree of freedom systems. Two multi-degree of freedom systems were examined, with two and six degrees of freedom, respectively. In both cases, the inputs to the systems are independent Gaussian white noise forces on each mass, and the output is the displacement of mass number 1.

Figure 39 demonstrates that the algorithm can identify and track a two degree of freedom system when one of the masses changes for a period of time. The frequencies and damping ratios for a time-invariant six degree of freedom system are shown in Fig. 40. It is worth noting that it takes a short amount of time for the estimators to settle on the correct values, but it does find them. The same observation can be seen in Fig. 41, which tracks the same system but with instantaneous jumps in one mass, causing changes in both frequency and damping. Here, it takes a short amount of time after the jumps for the estimator to settle back to the exact values, but it does appear able to do so.

Monitoring modal response of the SDOF system in the previous section is, of course, trivial since there is but one mode. Here, however, with a multi-degree of freedom system, the Kalman filter can be used to monitor modal responses. Figure 42 shows the modal responses for this time-varying 6DOF system. The accuracy of the estimation depends somewhat on how much each mode is excited and how much each mode contributes to the displacement of the measured output; Fig. 42 also shows that the portion of the output contributed by the higher frequency modes was significantly less than that by the lower frequency modes. Thus, it would be expected that the lower frequency modes would be better identified, and that is indeed the case.

The frequency content of the modes after a change in one of the masses, as monitored by the Kalman filter, for the 6DOF system is shown in Fig. 43. The top graph is the discrete Fourier transform of the modal responses over a short period and the bottom is an average of several of these DFTs. The expectation that lower frequency modes would be better observed and identified is obvious here, where the lowest modes demonstrate the least relative noise in the DFTs.



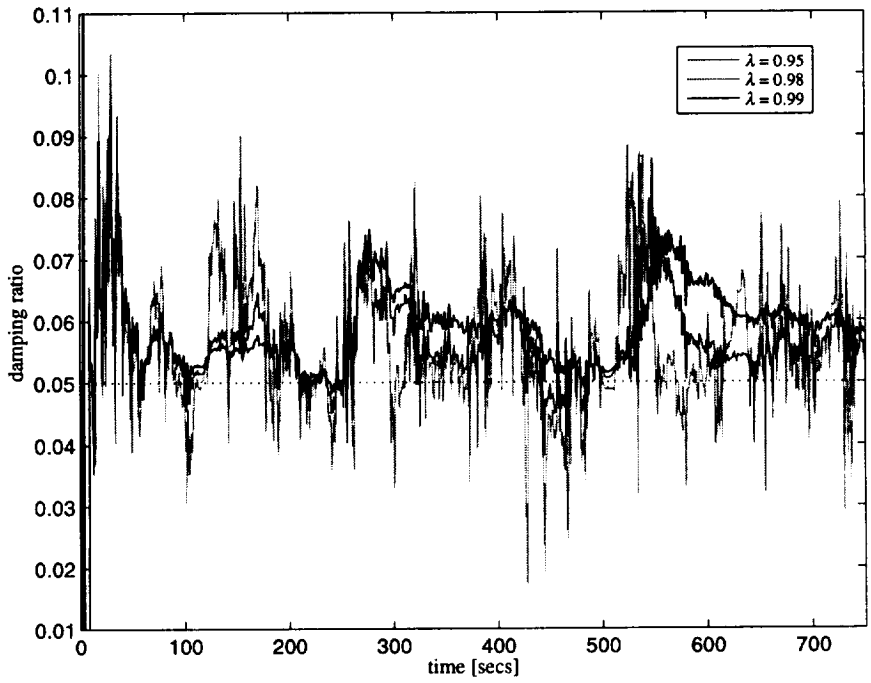


Figure 33: Tracking the frequency and damping of a piecewise time-invariant (spring stiffness decreasing) SISO SDOF system.

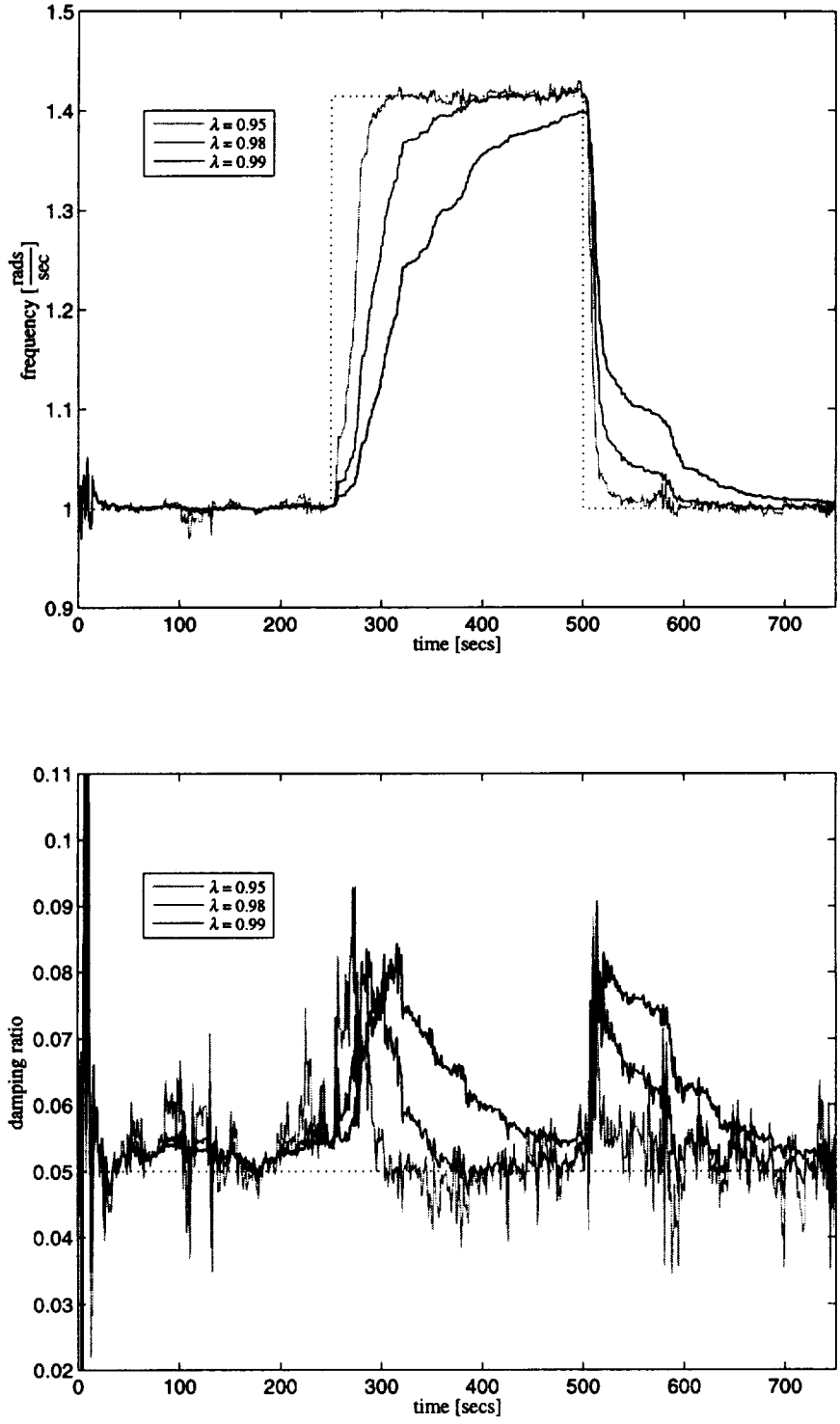


Figure 34: Tracking the frequency and damping of a piecewise time-invariant (spring stiffness increasing) SISO SDOF system.

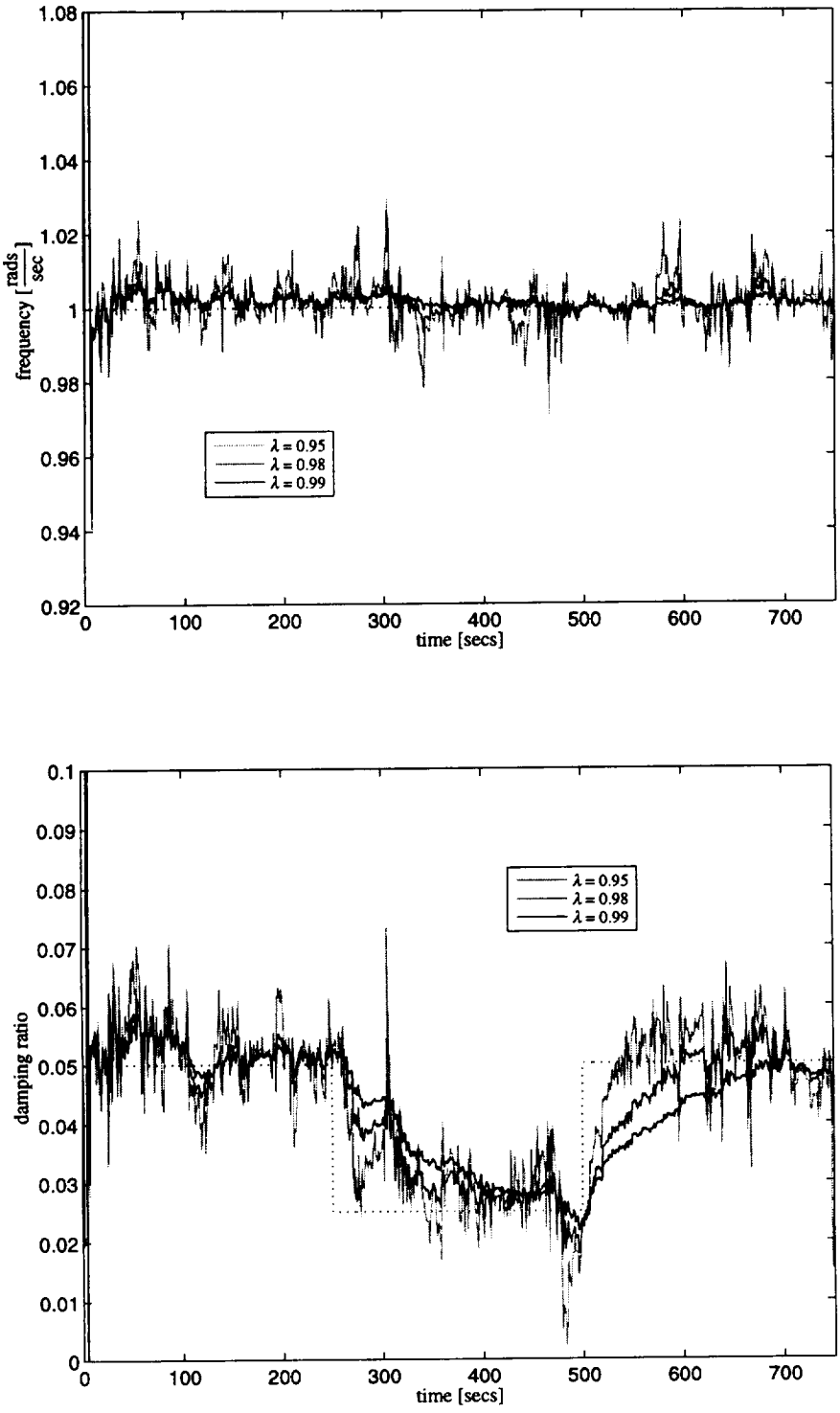


Figure 35: Tracking the frequency and damping of a piecewise time-invariant (damping decreasing) SISO SDOF system.

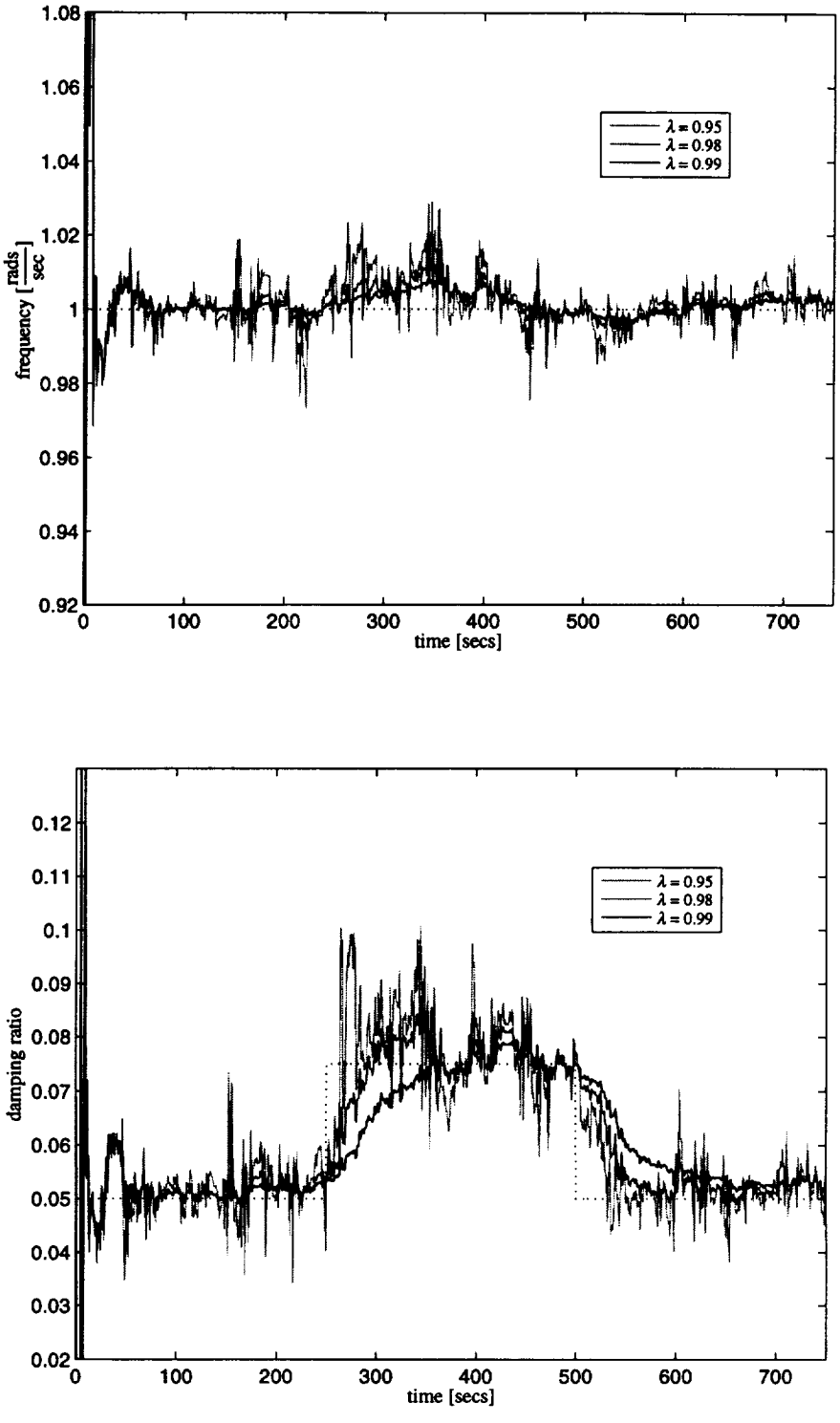


Figure 36: Tracking the frequency and damping of a piecewise time-invariant (damping increasing) SISO SDOF system.

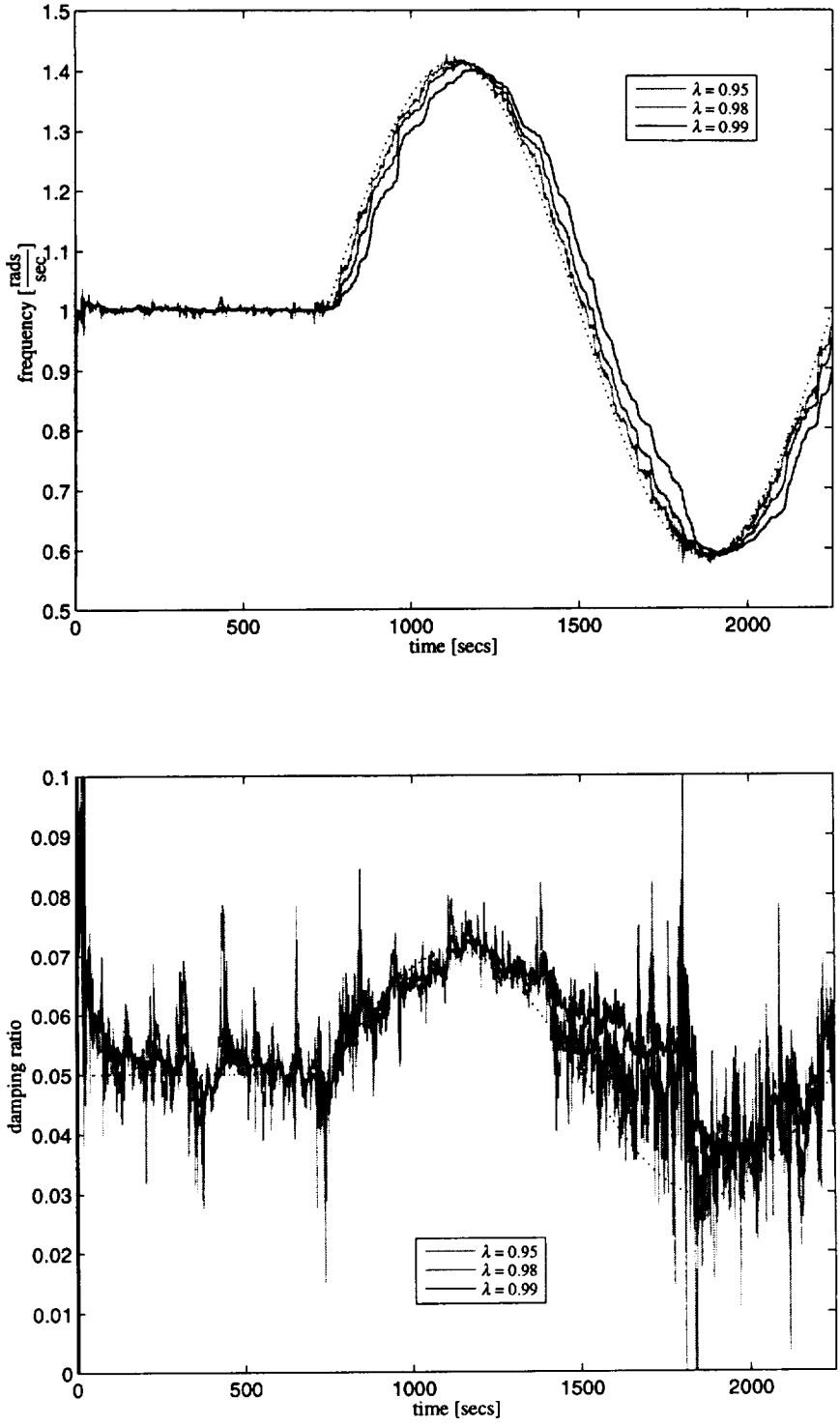
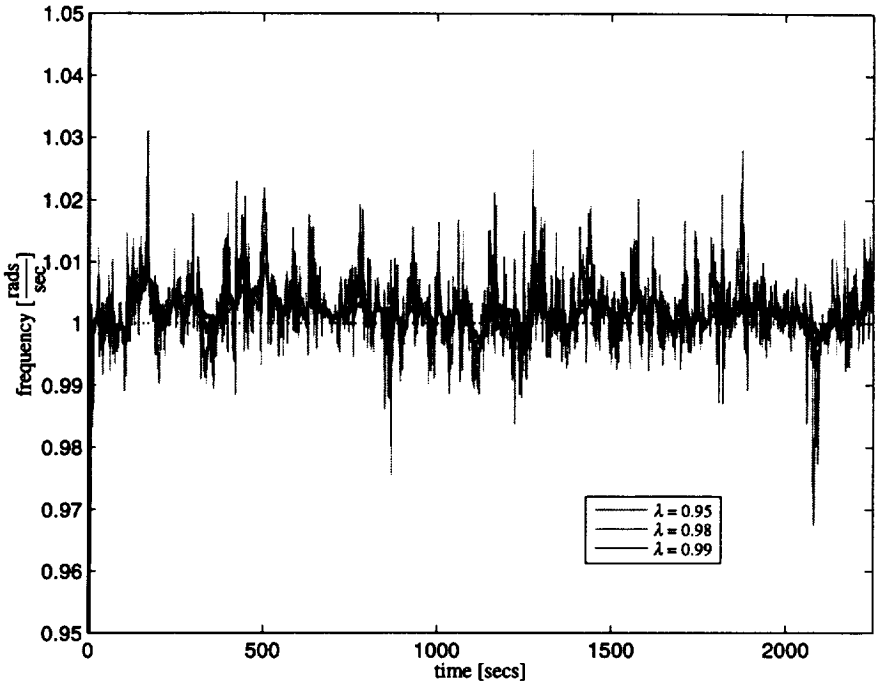


Figure 37: Tracking the frequency and damping of a continuously time-varying (mass changing) SISO SDOF system.



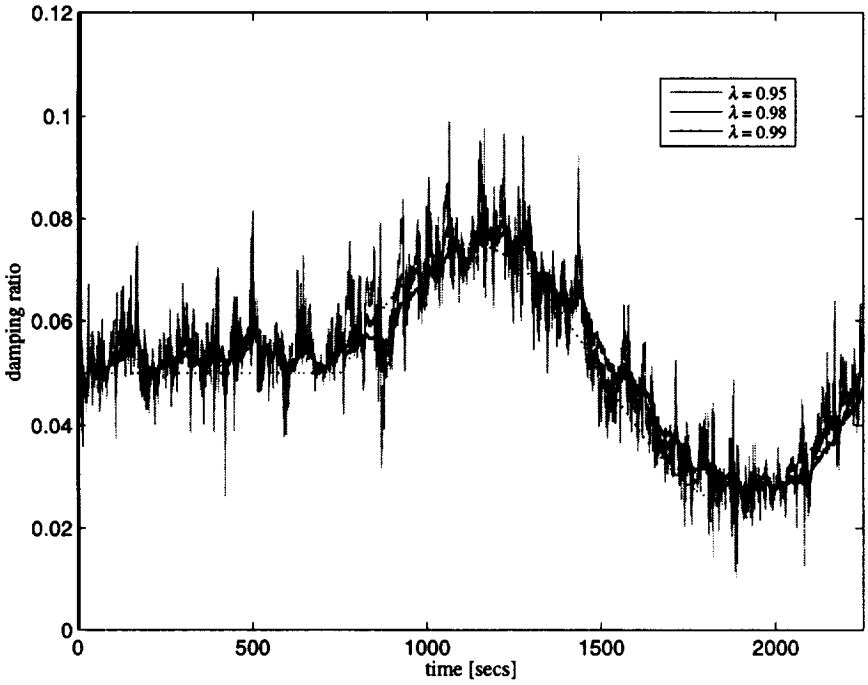


Figure 38: Tracking the frequency and damping of a continuously time-varying (damping changing) SISO SDOF system.

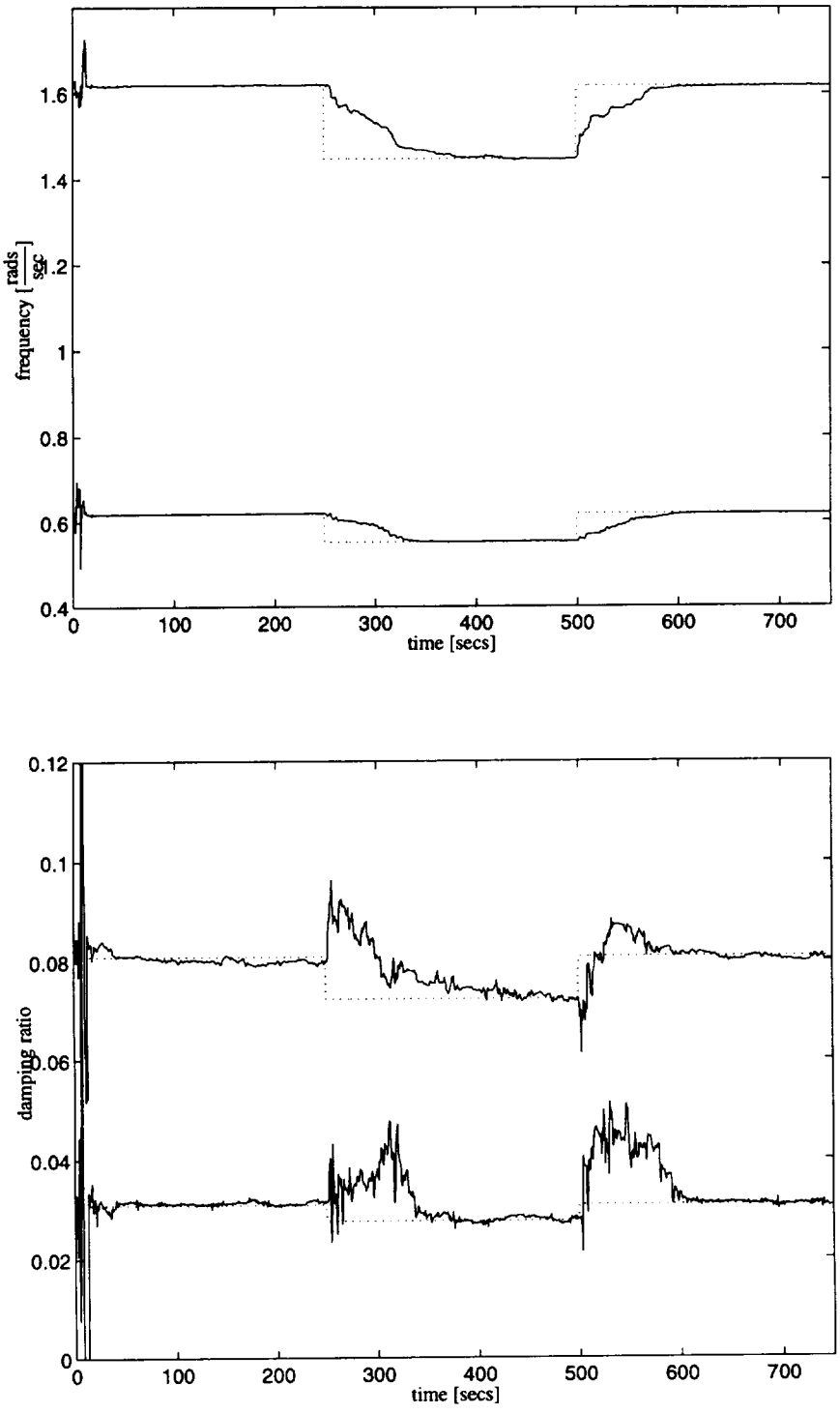


Figure 39: Tracking the frequency and damping of a piecewise time-invariant (one mass increases) MISO 2DOF system.

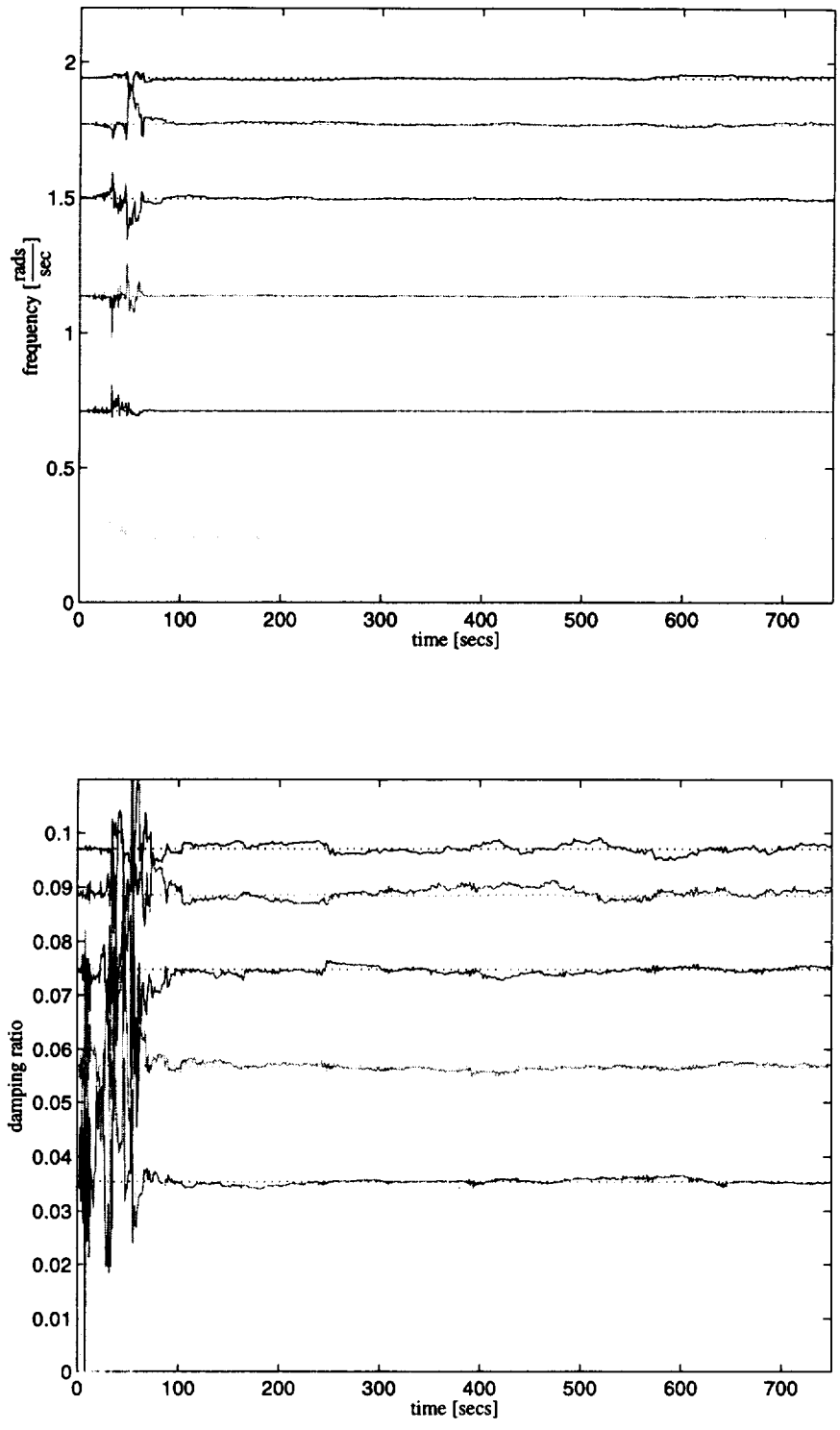


Figure 40: Tracking the frequency and damping of a time-invariant MISO 6DOF system.

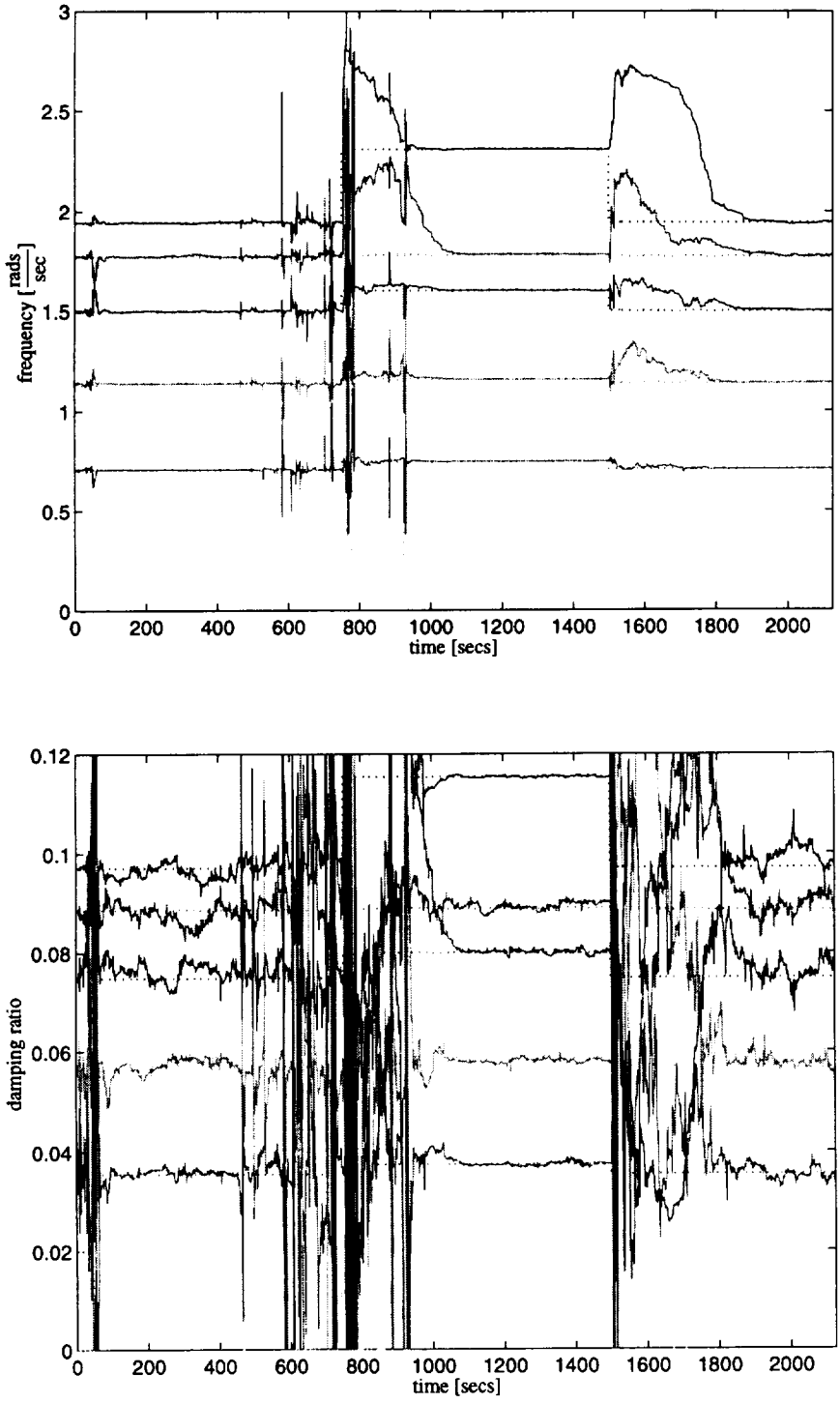


Figure 41: Tracking the frequency and damping of a piecewise time-invariant (mass #3 decreases) MISO 6DOF system.

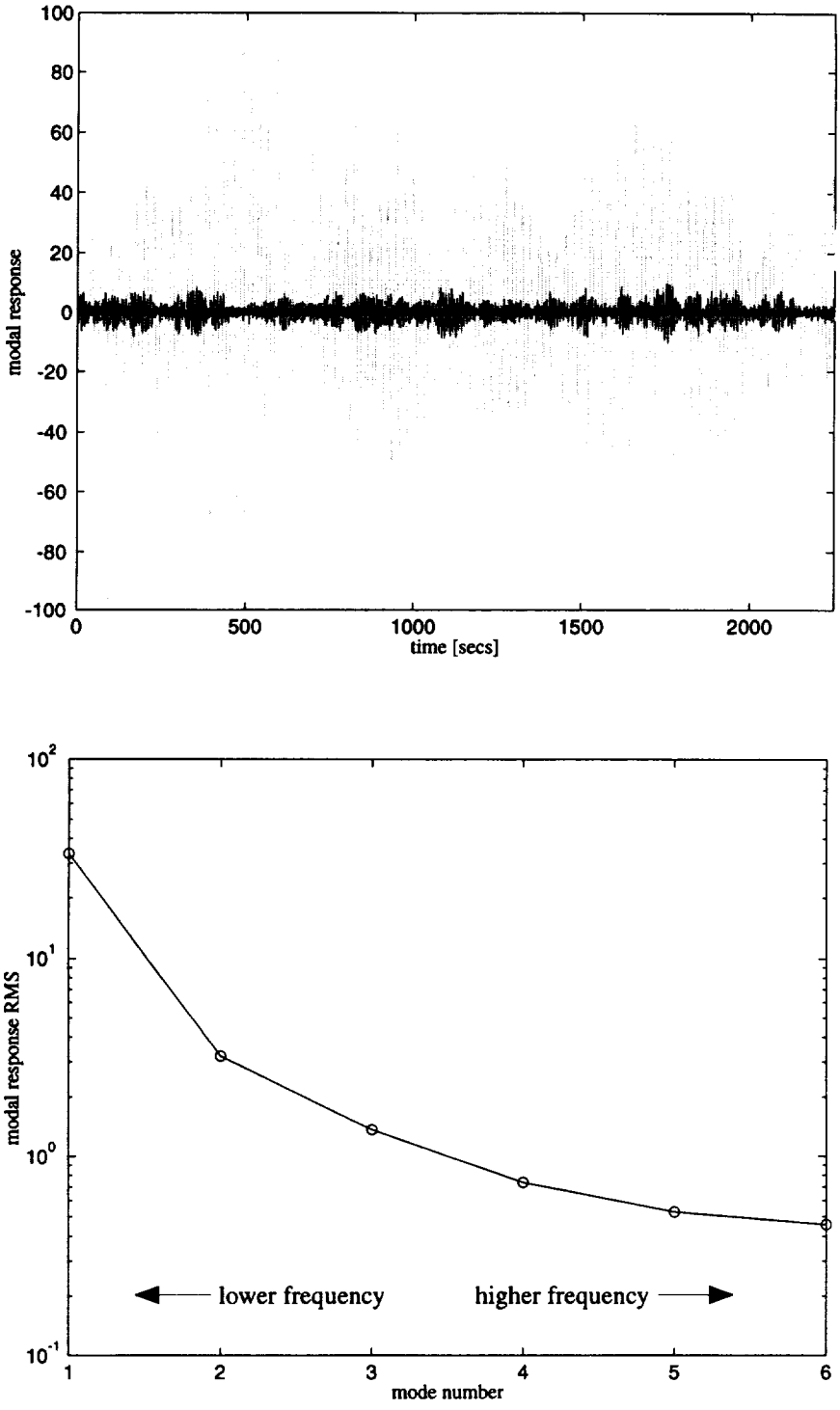


Figure 42: Modal response of MISO 6DOF system after one mass has decreased.

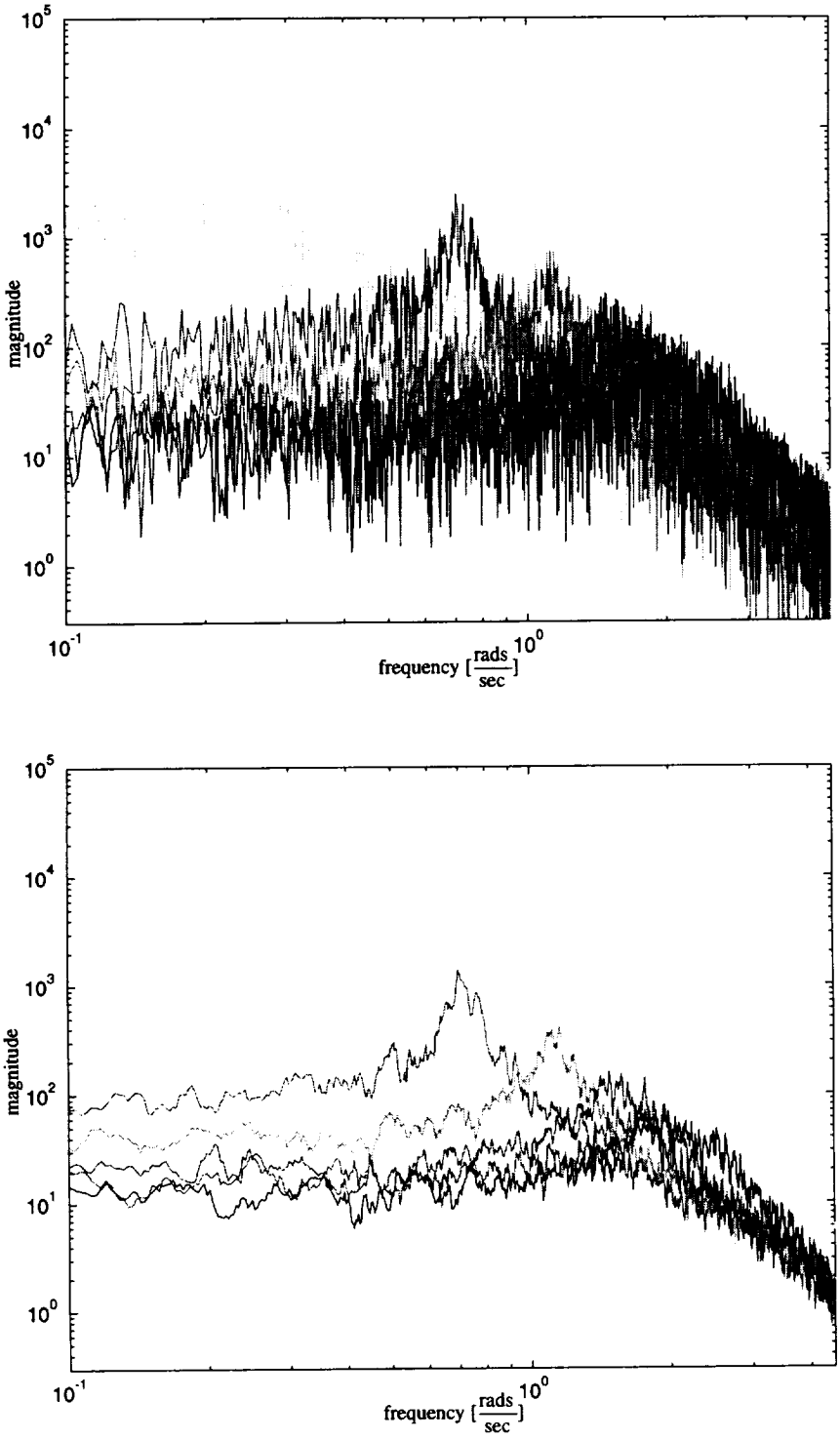


Figure 43: The frequency content of the 6 modal responses after one mass has decreased: one FFT (top) and several averaged FFTs (bottom).

5.2.4 Observations on Two-Stage Adaptive Monitoring

There are several important observations that can be made from the above examples. The first is that natural frequency estimation is consistently more robust than estimation of damping ratios. This observation is neither surprising nor unusual, since it is true for most identification methods. Nevertheless, damping estimates are available and do track with changes in system characteristics.

The `rarx` function in MATLAB® is relatively sensitive to high frequency noise, so the lowpass filter was found to be essential. Furthermore, it was seen that multi-degree of freedom systems demonstrated higher noise sensitivity than smaller systems. It is unclear whether these sensitivities are an artifact of the particular implementation of the recursive least-squares identification used in `rarx` or if it is inherent in the algorithm itself. From hints in the literature, it is suspected that it is the former.

The poor excitation of higher modes probably contributed to less accurate estimation of the higher frequencies and corresponding damping ratios. An example could, of course, be constructed to more evenly excite the various modes, and in which the output has similar contributions from the various modes; but that would, in some ways, be artificial, since one typically finds that modal contributions and modal excitation widely vary. Thus, the example shown here is the rule, not the exception.

6.0 CONCLUSIONS AND FUTURE DIRECTION

On-line monitoring of time-varying structural systems is a difficult problem. The Reciprocal Modal Vector method is useful for many systems but restricted to those whose modeshapes do not change over time. This is, of course, not the case most of the time. Furthermore, limited sensor arrays further disallow the use of the MRMV method.

H_∞-based identification methods work satisfactorily for small, off-line problems and they have been applied to a number of real-world problems. But they require a significant amount of interaction by the engineer. Furthermore, they also are relatively computationally intensive, requiring solutions of large eigenvalue problems, thus limiting their usefulness in an on-line context.

There are, however, a number of recursive identification algorithms that are useful in on-line monitoring. They can be implemented in a two-stage algorithm that also uses a modal Kalman filter to monitor modal responses in real-time.

Several issues remain for further study. No true multi-input, multi-output (MIMO) systems were studied here. The standard identification algorithms that are available within MATLAB® were used to study least-squares time-domain identification; the recursive versions of these algorithms are not implemented for MIMO systems in the current version of the *System Identification Toolbox* (Ljung, 1995). Furthermore, "fast" versions of the recursive least-squares methods need to be investigated for their claims regarding computational requirements being less than $O(n^2)$. This is necessary if large, complex systems are intended to be monitored.

7.0 REFERENCES

1. H. Akçay, G. Gu, and P.P. Khargonekar, 1992. "Identification in H_∞ with Nonuniformly Spaced Frequency Response Measurements." *1992 American Control Conference*, Chicago, Illinois, June 24-26, 1992. Proceedings (American Automatic Control Council, Evanston, Illinois, 1992), 246-250.

2. H. Akçay, G. Gu, and P.P. Khargonekar, 1993. "A Class of Algorithms for Identification in H_∞: Continuous-Time Case." *IEEE Transactions on Automatic Control*, **38**(2), Feb. 1993, 289-294.

3. V.M. Adamjan, D.Z. Arov, and M.G. Krein, 1971. "Analytic Properties of Schmidt Pairs for a Hankel Operator and the Generalized Schur-Takagi Problem." *Mathematics of the USSR – Sbornik*, **15**(1), Sept. 1971, 31-73 (Russian original Tom 86(128)).

4. J. Adcock, 1987. "Curve Fitter for Pole-Zero Analysis." *Hewlett-Packard Journal*, Jan. 1987, 33-36.

5. A.E. Aktan, V.J. Hunt, M.J. Lally, R.B. Stillmaker, D.L. Brown, and S.J. Shelley, 1995. "Field Laboratory for Modal Analysis and Condition Assessment of Highway Bridges." *13th International Modal Analysis Conference*, Nashville, Tennessee, Feb. 13-16, 1995. Proceedings (D.J. DeMichele director; Society for Experimental Mechanics, Bethel, CT, 1995), 718-727.

6. R.J. Allemang, 1980. "Investigation of Some Multiple Input/Output Frequency Response Function Experimental Modal Analysis Techniques." Ph.D. Dissertation, Department of Mechanical and Industrial Engineering, University of Cincinnati, 1980.

7. A. Bahri and A.J. Helmicki, 1995. "H_∞ Identification-Based Robust Control System Design." *1995 American Control Conference*, Seattle, Washington, June 21-23, 1995. Proceedings (American Automatic Control Council, Evanston, Illinois, 1995), 3556-3561.

8. E.-W. Bai and M.S. Andersland, 1994. "Stochastic and Worst Case System Identification Are Not Necessarily Incompatible." *Automatica*, **30**(9), Sept. 1994, 1491-1493.

9. E.-W. Bai and S. Raman, 1994. "Robust System Identification with Noisy Experimental Data: Projection Operator and Linear Algorithms." *Automatica*, **30**(7), July 1994, 1203-1206.

10. J.L. Beck, 1996. "System Identification Methods Applied to Measured Seismic Response." *11th World Conference on Earthquake Engineering*, Acapulco, Mexico, June 1996.

11. J.L. Beck and M.W. Vanik, 1996. "Structural Model Updating Using Expanded Mode-shapes." *11th ASCE Engineering Mechanics Specialty Conference*, Fort Lauderdale, Florida, May 19-22, 1996. Proceedings (Y.K. Lin and T.C. Su, eds., ASCE, New York, 1996), 152-155.

12. R.A. Canfield and L. Meirovitch, 1994. "Integrated Structural Design and Vibration Suppression Using Independent Modal Space Control." *AIAA Journal*, **32**(10), Oct. 1994, 2053-2060.

13. C.-W. Chen, J.-K. Huang, M. Phan, and J.-N. Juang, 1992. "Integrated System Identification and State Estimation for Control of Flexible Space Structures." *Journal of Guidance, Control, and Dynamics*, **15**(1), Jan.-Feb., 1992, 88-95.

14. J. Chen, G. Gu, and C.N. Nett, 1993. "Worst Case Identification of Continuous Time Systems via Interpolation." *1993 American Control Conference*, San Francisco, California, June 2-4, 1993. Proceedings (American Automatic Control Council, Evanston, Illinois, 1993), 1544-1548.

15. J. Chen, G. Gu, and C.N. Nett, 1994. "Worst Case Identification of Continuous Time Systems via Interpolation." *Automatica*, **30**(12), Dec. 1994, 1825-1837.

16. J. Chen and S. Wang, 1995. "New Time-Domain Algorithms for H_∞ Identification." *1995 American Control Conference*, Seattle, Washington, June 21-23, 1995. Proceedings (American Automatic Control Council, Evanston, Illinois, 1995), 1976-1980.

17. C.K. Chui and G. Chen, 1987. *Kalman Filtering with Real-Time Applications.* Springer-Verlag (Berlin), 1987.

18. J. Cioffi and T. Kailath, 1984. "Fast Recursive Least-Squares Transversal Filters for Adaptive Filtering." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2), 304-337.

19. M.A. Dahleh and I.J. Diaz-Bobillo, 1995. *Control of Uncertain Systems: A linear programming approach.* Prentice Hall (Englewood Cliffs, NJ), 1995.

20. G. Didinsky, Z. Pan, and T. Basar, 1995. "Parameter Identification for Uncertain Plants using H_∞ Methods." *Automatica*, **31**(9), Sept. 1995, 1227-1250.

21. L.C. Freudinger, 1990. "Analysis of Structural Response Data Using Discrete Modal Filters." M.S. thesis, Department of Mechanical, Industrial, and Nuclear Engineering, University of Cincinnati, Cincinnati, Ohio, 1990.

22. L.C. Freudinger, 1991. "Analysis of Structural Response Data Using Discrete Modal Filters." NASA Contractor Report CR-179448, May 1991.

23. J.H. Friedman, 1996. "Identification, Modeling, and Control of Flexible Structures." Ph.D. dissertation, Department of Aerospace Engineering, University of Michigan, Ann Arbor, Michigan, 1996.

24. J.H. Friedman and P.P. Khargonekar, 1995a. "A Comparative Applications Study of Frequency Domain Identification Techniques." *1995 American Control Conference*, Seattle, Washington, June 21-23, 1995. Proceedings (American Automatic Control Council, Evanston, Illinois, 1995), 3055-3059.

25. J.H. Friedman and P.P. Khargonekar, 1995b. "Application of Identification in H_∞ to Lightly Damped Systems: two case studies." *IEEE Transactions on Control Systems Technology*, **3**(3), Sept. 1995, 279-289.

26. R. Ghanem and M. Shinozuka, 1995. "Structural System Identification, I: Theory." *Journal of Engineering Mechanics*, **121**(2), Feb. 1995, 255-264.

27. L.J. Griffiths, 1977. "A Continuously Adaptive Filter Implemented as a Lattice Structure." *1977 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Hartford, Connecticut, May 9-11, 1977. Proceedings (IEEE, Piscataway, NJ, 1977), 683-686.

28. G. Gu, C.-C. Chu, and G. Kim, 1994. "Linear Algorithms for Worst Case Identification in H_∞ with Applications to Flexible Structures." *1994 American Control Conference*, Baltimore, Maryland, June 29 - July 1, 1994. Proceedings (American Automatic Control Council, Evanston, Illinois, 1994), 112-116.

29. G. Gu and P.P. Khargonekar, 1991. "Linear and Nonlinear Algorithms for Identification in H_∞ with Error Bounds." *1991 American Control Conference*, Boston, Massachusetts, June 26-28, 1991. Proceedings (American Automatic Control Council, Green Valley, Arizona, 1991), 64-69.

30. G. Gu and P.P. Khargonekar, 1992a. "A Class of Algorithms for Identification in H_∞." *Automatica*, **28**(2), March 1992, 299-312.

31. G. Gu and P.P. Khargonekar, 1992b. "Linear and Nonlinear Algorithms for Identification in H_∞ with Error Bounds." *IEEE Transactions on Automatic Control*, **37**(7), July 1992, 953-963.

32. G. Gu and P.P. Khargonekar, 1993. "Frequency Domain Identification of Lightly Damped Systems: The JPL Example." *1993 American Control Conference*, San Francisco, California, June 2-4, 1993. Proceedings (American Automatic Control Council, Evanston, Illinois, 1993), 3052-3056.

33. G. Gu, P.P. Khargonekar, and E.B. Lee, 1989. "Approximation of Infinite Dimensional Systems." *IEEE Transactions on Automatic Control*, **34**(6), June 1989, 610-618.

34. A.J. Helmicki, C.A. Jacobson, and C.N. Nett, 1989. "H_∞ Identification of Stable LSI Systems: A Scheme with Direct Application to Controller Design." *1989 American Control Conference*, Pittsburgh, Pennsylvania, June 21-23, 1989. Proceedings (American Automatic Control Council, Green Valley, Arizona, 1989), 1428-1434.

35. A.J. Helmicki, C. A. Jacobson, and C.N. Nett, 1990a. "Identification in H_∞: A Robustly Convergent, Nonlinear Algorithm." *1990 American Control Conference*, San Diego, California, May 23-25, 1990. Proceedings (American Automatic Control Council, Green Valley, Arizona, 1990), 386-391.

36. A.J. Helmicki, C.A. Jacobson, and C.N. Nett, 1990b. "Identification in H_∞: The Continuous-Time Case." *1990 American Control Conference*, San Diego, California, May 23-25, 1990. Proceedings (American Automatic Control Council, Green Valley, Arizona, 1990), 1893-1898.

37. A.J. Helmicki, C.A. Jacobson, and C.N. Nett, 1990c. "Identification in H_∞: Linear Algorithms." *1990 American Control Conference*, San Diego, California, May 23-25, 1990. Proceedings (American Automatic Control Council, Green Valley, Arizona, 1990), 2418-2423.

38. A.J. Helmicki, C.A. Jacobson, and C.N. Nett, 1991a. "Fundamentals of Control-Oriented System Identification and Their Application for Identification in H_∞." *1991 American Control Conference*, Boston, Massachusetts, June 26-28, 1991. Proceedings (American Automatic Control Council, Green Valley, Arizona, 1991), 89-99.

39. A.J. Helmicki, C.A. Jacobson, and C.N. Nett, 1991b. "Control Oriented System Identification: A Worst-Case Deterministic Approach in H_∞." *IEEE Transactions on Automatic Control*, **36**(10), Oct. 1991, 1163-1176.

40. A.J. Helmicki, C.A. Jacobson, and C.N. Nett, 1992. "Worst-Case Deterministic Identification in H_∞: The Continuous-Time Case." *IEEE Transactions on Automatic Control*, **37**(5), May 1992, 604-610.

41. D.J. Inman, 1989. *Vibration: with Control, Measurement, and Stability.* Prentice Hall (Englewood Cliffs, NJ), 1989.

42. C.A. Jacobson and G. Tadmor, 1993. "A Note on H_∞ System Identification With Probabilistic Apriori Information." *1993 American Control Conference*, San Francisco, California, June 2-4, 1993. Proceedings (American Automatic Control Council, Evanston, Illinois, 1993), 1539-1543.

43. R.N. Jacques, 1994. "On-line System Identification and Control Design for Flexible Structures." Ph.D. dissertation, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Mass., 1994.

44. E.A. Johnson, L.A. Bergman, P.G. Voulgaris, and L.C. Freudinger, 1996. "Modal Filter Based On-line Monitoring of Uncertain Structural Systems." *11th ASCE Engineering Mechanics Specialty Conference*, Fort Lauderdale, Florida, May 19-22, 1996. Proceedings (Y.K. Lin and T.C. Su, eds., ASCE, New York, 1996), 156-159.

45. J.-N. Juang, 1994. *Applied System Identification.* Prentice Hall (Englewood Cliffs, NJ), 1994.

46. J.-N. Juang and R.S. Pappa, 1985. "An Eigensystem Realization Algorithm for Modal Parameter Identification and Model Reduction." *Journal of Guidance, Control, and Dynamics*, **8**(5), Sept.-Oct. 1985, 620-627.

47. J.-N. Juang, M. Phan, L.G. Horta, and R.W. Longman, 1993. "Identification of Observer/ Kalman Filter Markov Parameters: Theory and Experiments." *Journal of Guidance, Control, and Dynamics*, **16**(2), March-April 1993, 320-329.

48. R.L. Kosut, G.C. Goodwin, and M.P. Polis, 1992. "Introduction, Special Issue on System Identification for Robust Control Design." *IEEE Transactions on Automatic Control*, **37**(7), July 1992, 899.

49. D.T. Lee, M. Morf, and B. Friedlander, 1981. "Recursive Least Squares Ladder Estimation Algorithms." *IEEE Transactions on Acoustics, Speech,* and Signal Processing, 29(3), 627-641.

50. E. Levy, 1959. "Complex-Curve Fitting." *IRE Transactions on Automatic Control*, **AC-4**(1), May 1959, 37-44.

51. R. Lind and M. Brenner, 1996. "μ-Based Robust Stability Estimation of Aeroelastic Systems using Flight Derived Uncertainty Models." Submitted to the *AIAA Journal of Guidance, Control and Dynamics*, 1996.

52. K. Liu and R. Skelton, 1993. "Q-Markov Covariance Equivalent Realization and its Application to Flexible Structure Identification." *Journal of Guidance, Control, and Dynamics*, **16**(2), Mar.-Apr. 1993, 308-319.

53. L. Ljung, 1987. *System Identification: Theory for the user.* Prentice Hall (Englewood Cliffs, NJ), 1987.

54. L. Ljung, 1995. *System Identification Toolbox 4.0 User's Guide.* The MathWorks, Inc. (Natick, Mass.), 1995.

55. L. Ljung and T. Soderstrom, 1983. *Theory and Practice of Recursive Identification.* MIT Press (Cambridge, Mass.), 1983.

56. J. Makhoul, 1977. "Stable and Efficient Lattice Methods for Linear Prediction." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **25**(5), 423-428.

57. P.M. Mäkilä, 1991a. "Laguerre Methods and H_∞ Identification of Continuous-Time Systems." *International Journal of Control*, **53**(3), March 1991, 689-707.

58. P.M. Mäkilä, 1991b. "Robust Identification and Galois Sequences." *International Journal of Control*, **54**(5), Nov. 1991, 1189-1200.

59. P.M. Mäkilä, 1992. "Worst-Case Input-Output Identification." *International Journal of Control*, **56**(3), Sept. 1992, 673-689.

60. P.M. Mäkilä, 1993. "Robust Approximate Modeling from Noisy Point Evaluations." *1993 American Control Conference*, San Francisco, California, June 2-4, 1993. Proceedings (American Automatic Control Council, Evanston, Illinois, 1993), 1554-1560.

61. P.M. Mäkilä and J.R. Partington, 1991. "Robust Approximation and Identification in H_∞." *1991 American Control Conference*, Boston, Massachusetts, June 26-28, 1991. Proceedings (American Automatic Control Council, Green Valley, Arizona, 1991), 70-76.

62. P.M. Mäkilä and J.R. Partington, 1992a. "Worst-Case Identification from Closed-Loop Time Series." *1992 American Control Conference*, Chicago, Illinois, June 24-26, 1992. Proceedings (American Automatic Control Council, Evanston, Illinois, 1992), 301-306.

63. P.M. Mäkilä and J.R. Partington, 1992b. "Robust Identification of Strongly Stabilizable Systems." *IEEE Transactions on Automatic Control*, **37**(11), Nov. 1992, 1709-1716.

64. P.M. Mäkilä and J.R. Partington, 1993. "Robust Approximate Modelling of Stable Linear Systems." *International Journal of Control*, **58**(3), Sept. 1993, 665-683.

65. P.M. Mäkilä, J.R. Partington, and T. K. Gustafsson, 1995. "Worst-case Control-relevant Identification." *Automatica*, **31**(12), Dec. 1995, 1799-1819.

66. L. Meirovitch and H. Baruh, 1982. "Control of Self-Adjoint Distributed-Parameter Systems." *Journal of Guidance, Control, and Dynamics*, **5**(1), Jan.-Feb. 1982, 60-66.

67. L. Meirovitch and H. Baruh, 1985. "The Implementation of Modal Filters for Control of Structures." *Journal of Guidance, Control, and Dynamics*, **8**(6), Nov.-Dec. 1985, 707-716.

68. L. Meirovitch and D. Ghosh, 1987. "Control of Flutter in Bridges." *Journal of Engineering Mechanics*, **113**(5), May 1987, 720-736.

69. K.M. Nagpal and P.P. Khargonekar, 1991. "Filtering and Smoothing in an H_∞ Setting," *IEEE Transactions on Automatic Control*, **36**(2), Feb. 1991, 152-166.

70. Z. Nehari, 1957. "On Bounded Bilinear Forms." *Annals of Mathematics*, **65**(1), Jan. 1957, 153-162.

71. H. Öz and L. Meirovitch, 1983. "Stochastic Independent Modal-Space Control of Distributed-Parameter Systems." *Journal of Optimization Theory and Applications*, **40**(1), May 1983, 121-154.

72. H. Öz and L. Meirovitch, 1984. "Digital Stochastic Control of Distributed-Parameter Systems." *Journal of Optimization Theory and Applications*, **43**(2), June 1984, 307-325.

73. P.J. Parker and R.R. Bitmead, 1987a. "Adaptive Frequency Response Identification." *26th IEEE Conference on Decision and Control*, Los Angeles, California, December 9-11, 1987. Proceedings (IEEE, New York, 1987), 348-353.

74. P.J. Parker and R.R. Bitmead, 1987b. "Approximation of Stable and Unstable Systems via Frequency Response Identification." *10th IFAC World Congress*, Munich, Germany, July 27-31, 1987. Proceedings (R. Isermann, ed., IFAC/Pergamon, Oxford, England, 1988), Volume X, 358-363.

75. J.R. Partington, 1991. "Robust Identification and Interpolation in H_∞." *International Journal of Control*, **54**(5), Nov. 1991, 1281-1290.

76. J.R. Partington, 1992. "Robust Identification in H_∞." *Journal of Mathematical Analysis and Applications*, **166**, 1992, 428-441.

77. J.R. Partington, 1993. "Algorithms for Identification in H_∞ with Unequally Spaced Function Measurements." *International Journal of Control*, **58**(1), July 1993, 21-31.

78. J.R. Partington and P.M. Mäkilä, 1995a. "Worst-Case Analysis of the Least-Squares Method and Related Identification Methods." *Systems and Control Letters*, **24**(3), Feb 13., 1995, 193-200.

79. J.R. Partington and P.M. Mäkilä, 1995b. "Analysis of Linear Methods for Robust Identification in l_1." *Automatica*, **31**(5), May 1995, 755-758.

80. L.D. Peterson, 1993. "Efficient Computation of the Eigensystem Realization Algorithm." Technical Report CU-CSS-93-7, Center for Space Structures and Engineering, University of Colorado, Boulder, Colorado, April 1993.

81. L.D. Peterson, 1995. "Efficient Computation of the Eigensystem Realization Algorithm." *Journal of Guidance, Control, and Dynamics*, **18**(3), May-June 1995, 395-403.

82. K. Poola, M. Safonov, and R. Smith, 1993. "Robust Identification and Control Tutorial Workshop," at an IEEE Aerospace Conference, May 27, 1993.

83. B. Priel, E. Soroka, and U. Shaked, 1991. "The Design of Optimal Reduced-Order Stochastic Observers for Discrete-Time Linear Systems." *IEEE Transactions on Automatic Control*, **36**(11), Nov. 1991, 1300-1307. Reprinted in **36**(12), Dec. 1991, 1502-1509.

84. C. Sanathanan and J. Koerner, 1963. "Transfer Function Synthesis as a Ratio of Two Complex Polynomials." *IEEE Transactions on* Automatic Control, 8(1), Jan. 1963, 56-58.

85. J.F. Schultze, R.W. Rost, and S.J. Shelly, 1996. "Adaptive Modal Space Control of Flexible Structures: Theory." *14th International Modal Analysis Conference*, Dearborn, Michigan, Feb. 12-15, 1996. Proceedings (A.L. Wicks, technical program chair; Society for Experimental Mechanics, Bethel, CT, 1996), 292-299.

86. S.J. Shelley, 1991. "Investigation of Discrete Modal Filters for Structural Dynamic Applications." Ph.D. dissertation, Department of Mechanical, Industrial, and Nuclear Engineering, University of Cincinnati, Cincinnati, Ohio, 1991.

87. S.J. Shelley, A.E. Aktan, D.L. Brown, and R.J. Allemang, 1994. "University of Cincinnati Experience with Implementing Active Structural Vibration Control." *First World Conference on Structural Control*, Los Angeles, CA, August 3-5, 1994. Proceedings, FP4–62-70.

88. S.J. Shelley, A.E. Aktan, and K.L. Lee, 1994. "Modal Filter Based Structural Control of a Highway Bridge." *Structures Congress '94, 11th ASCE Conference on Analysis and Computation*, Atlanta, Georgia, April 24-26, 1994. Proceedings (N.C. Baker and B.J. Goodno, eds., ASCE, New York, 1994), 347-356.

89. S.J. Shelley and R.J. Allemang, 1994. "An Investigation of Online Vibration Parameter Estimation Schemes." NASA-Ames University Consortium, Number NCA2-734, 1994.

90. S.J. Shelley, R.J. Allemang, G.L. Slater, and J.F. Schultze, 1993. "Active Vibration Control Utilizing an Adaptive Modal Filter Based Modal Control Method." *11th International Modal Analysis Conference*, Kissimmee, Florida, Feb. 1-4, 1993. Proceedings (D.J. DeMichele director; Society for Experimental Mechanics, Bethel, CT, 1993), 751-758.

91. S.J. Shelley, L.C. Freudinger, and R.J. Allemang, 1993. "Development of an On-line Modal State Monitor." *11th International Modal Analysis Conference*, Kissimmee, Florida, Feb. 1-4, 1993. Proceedings (D.J. DeMichele, director; Society for Experimental Mechanics, Bethel, CT, 1993), 606-612.

92. S.J. Shelley, K.L. Lee, T. Aksel, and A.E. Aktan, 1995. "Active-Control and Forced-Vibration Studies on Highway Bridge." *Journal of Structural Engineering*, **121**(9), Sept. 1995, 1306-1312.

93. M. Shinozuka and R. Ghanem, 1995. "Structural System Identification, II: Experimental Verification." *Journal of Engineering Mechanics*, **121**(2), Feb. 1995, 265-273.

94. T. Söderström and K.J. Åström, 1995. "Special Issue on Trends in System Identification." *Automatica*, **31**(12), Dec. 1995, 1689-1690.

95. P. Van Overschee and B. De Moor, 1994. "N4SID: Subspace Algorithms for the Identification of Combined Deterministic-Stochastic Systems." *Automatica*, **30**(1), Jan. 1994, 75-93.

96. P. Van Overschee and B. De Moor, 1995. "A Unifying Theorem for Three Subspace System Identification Algorithms." *Automatica*, **31**(12), Dec. 1995, 1853-1864.

97. M. Viberg, 1995. "Subspace-based Methods for the Identification of Linear Time-invariant Systems." *Automatica*, **31**(12), Dec. 1995, 1835-1851.

98. J.C. Wei, R.J. Allemang, and Y.W. Luk, 1994. "Direct Updating Based on Modal Filter Vectors." *12th International Modal Analysis Conference*, Honolulu, Hawaii, Jan. 31 - Feb. 3, 1994. Proceedings (D.J. DeMichele *et al.*, eds., Society for Experimental Mechanics, Bethel, CT, 1994), 219-224.

99. I. Yaesh and U. Shaked, 1992. "Game Theory Approach to Optimal Linear State Estimation and its Relation to the Minimum H_∞-Norm Estimation." *IEEE Transactions on Automatic Control*, **37**(6), June 1992, 828-831.

100. N. Young, 1988. *An Introduction to Hilbert Space*, Cambridge University Press (Cambridge), 1988.

101. G. Zames, 1979. "On the Metric Complexity of Causal Linear Systems: ε-entropy and ε-dimension for Continuous-Time." *IEEE Transactions on Automatic Control*, **24**(2), April 1979, 222-230.

102. Q. Zhang, R.J. Allemang, and D.L. Brown, 1990. "Modal Filter: Concept and Applications." *8th International Modal Analysis Conference*, Orlando, Florida, Jan. 1990. Proceedings (Society for Experimental Mechanics, Bethel, CT, 1990), 487-496.

8.0 APPENDIX A: COMPUTER CODES

The codes that follow are all MATLAB® functions and scripts. They are available from the authors by e-mailing to `johnsone@uiuc.edu`. Some of the example scripts require the MATLAB® `stextfun` toolbox (written by Douglas M. Schwarz <schwarz@kodak.com> and available at `ftp://ftp.mathworks.com/pub/contrib/graphics/stextfun`) to add styled text on graphs. A few of the codes here may require some of functions in the standard toolboxes (*e.g.*, the Control System Toolbox); `mrmvtool`, the GUI front end to `mrmv`, requires the `UITools` toolbox from The MathWorks.

8.1 MRMV CODES

8.1.1 `mrmv.m` — Modified Reciprocal Modal Vector

`mrmv` is the primary code to compute Reciprocal Modal Vectors. Its input and output arguments are explained in the help section of the code.

```
function [rmv,mpv,ufrf,robind,condnum,err_calc,err_total,cor_calc,cor_total, ...
          warn] = mrmv(l,w,resp,ni,w_index,out_order,realonly,returnerror)
% MRMV computes the Reciprocal Modal Vectors for the given system response.
%
%   [RMV,MPV,UFRF,ROBIND,CONDNUM,ERR_CALC,ERR_TOTAL,COR_CALC,COR_TOTAL]
%      = MRMV(L,W,RESP,NUMINPUTS,W_INDEX,OUT_ORDER,REALONLY)
%
%   Inputs:  L          the complex roots of the modes of interest
%                       (only one of each complex pair should be supplied)
%            W          a vector of frequencies in RADS/SEC
%            RESP       frequency response of the system; each column is
%                       the transfer function between an input and an output
%                       over the frequencies in W; the column in which is
%                       found a given frf is ((output#)+(input#-1)*(#outputs))
%            NUMINPUTS (optional) The number of inputs used (default=1)
%            W_INDEX    (optional) index of frequencies to use in calculating
%                       the RMVs (this is an option that permits the use of
%                       fewer points in the frf in order to reduce computational
%                       expense and focus on less noisy or more important
%                       frequency ranges)
%                       Note: an empty matrix is the same as choosing all freqs.
%            OUT_ORDER (optional) 2 ==> response is acceleration data
%                                 1 ==> response is velocity data
%                       (default) 0 ==> response is displacement data
%            REALONLY  (optional) 1 ==> restrict RMVs to be real (non-complex)
%                       (default) 0 ==> don't restrict them
%
%   Outputs:  RMV        matrix of reciprocal modal vectors (column by column)
%             MPV        matrix of modal participation factors
%             UFRF       matrix of uncoupled frfs; column number
%                        (mode#)+(input#-1)*(#modes) is the ufrf of mode
%                        (mode#) due to input at input (input#).  An additional
%                        (#modes) columns at the end using the MPV as a right
%                        weighting vector are used if multiple inputs are used.
%             ROBIND     matrix of modal filter robustness indicators;
%                        #cols = #modes, #rows = #outputs
%             CONDNUM    a row vector of the condition numbers of the RMVs
%                        (each element is the euclidian norm of the
%                        corresponding column of ROBIND)
%
%             ERR_CALC   least square error in uncoupled freq resp functions
%                        at the frequencies of interest, W(W_INDEX)
%             ERR_TOTAL  as ERR_CALC but at all frequencies W
%             COR_CALC   correlation in uncoupled freq resp functions
%                        at the frequencies of interest, W(W_INDEX)
%             COR_TOTAL  as COR_CALC but at all frequencies W
%
%   Requires NORMV, SBYS2STACK, and STACK2SBYS.
```

mrmv.m — Modified Reciprocal Modal Vector (cont.)

```
% Original MRMV code by Stuart J. Shelley, 1991, in his Ph.D. disseration.
% But it has overconfusing options and too much user interaction to be
%   efficient, so I've rewritten it completely.

% Copyright (c)1994-6, Erik A. Johnson <johnsone@uiuc.edu>
%  10/22/94 EAJ Original rewrite.
%   3/22/95 EAJ Fixed several minor bugs.
%               Increased efficiency (e.g., replaced pinv(A)*B with A\B).
%  11/03/95 EAJ Replaced several outdated auxiliary routines.
%   1/10/96 EAJ Fixed optional arguments so that they may be passed as [].
%               Clarified error messages.
%   5/15/96 EAJ Fixed index problem with ni==1.
%               Allowed W_INDEX to be a mask or indices.

% check number of inputs and outputs
if (nargin<3), error('MRMV requires at least three arguments:  L, W, RESP.');
elseif (nargin>8), error('MRMV takes at most 8 input arguments.');
elseif (nargout>10), error('MRMV produces at most 10 outputs.');
end;

% make w a column vector and l a row vector
w=w(:);
l=(l(:)).';

% create optional input arguments
if (nargin<=3), ni=[]; end;
if (nargin<=4), w_index=[]; end;
if (nargin<=5), out_order=[]; end;
if (nargin<=6), realonly=[]; end;
if (nargin<=7), returnerror=0; end;
warn = '';

% check input argument sizes
w_index=w_index(:);
if (~all(size(ni))), ni=1; end;
if (isempty(w_index)),
    w_index=(1:length(w))';
elseif (any(w_index>length(w))|(any(w_index<1)&(length(w_index)~=length(w)))),
    errmsg = 'MRMV got bad frequency indices.';
    if (returnerror), rmv=errmsg; return; else, error(errmsg); end;
end;
if (size(resp,1)~=length(w)),
    errmsg = ['MRMV requires that the #of rows in the '...
              'frfs & the #of frequencies be the same.'];
    if (returnerror), rmv=errmsg; return; else, error(errmsg); end;
elseif (rem(size(resp,2),ni)~=0),
    errmsg = 'MRMV requires that #of columns of RESP be a multiple of NUMINPUTS.';
    if (returnerror), rmv=errmsg; return; else, error(errmsg); end;
end;
if (~all(size(out_order))),
    out_order=0;
elseif isstr(out_order),
    ii = find(lower(out_order(1))=='dva');
    if (all(size(ii))),
        out_order = ii-1;
    else,
        errmsg = ['MRMV does not recognize ''' out_order(:).' ...
                  ''' as a valid OUT_ORDER.'];
        if (returnerror), rmv=errmsg; return; else, error(errmsg); end;
    end;
end;
if (~all(size(realonly))), realonly=0; else, realonly=realonly(1); end;

% just use the frequencies given by the index
if (all(w_index==0|w_index==1) & length(w_index)==size(resp,1)),
    % we got a mask; convert to indices
    w_index = find(w_index);
end;
w_allw=w;
w=w_allw(w_index);
```

mrmv.m — Modified Reciprocal Modal Vector (cont.)

```
% get sizes of things
nw=length(w);
no=size(resp,2)/ni;
nl=length(l);

% just use resp values corresponding to frequencies given by w_index
% stack the resp values
resp_allw=resp;
H=[sbys2stack(resp_allw(w_index,:),size(resp_allw,2)/no) zeros(nw*ni,ni-1)];

% create rhs vector
D = zeros(nw*ni,1);
lc=conj(l);
ww=w(:,ones(1,nl));
Di = 1./(ww+lc(ones(nw,1),:)*sqrt(-1)) - 1./(ww+l(ones(nw,1),:)*sqrt(-1));
if (out_order~=0), Di=Di.*((ww*sqrt(-1)).^out_order); end;
clear('ww');
if (ni>1),
    Di_index = (1:nw)';
    Di_index = Di_index(:,ones(1,ni-1));
    H_index = nw*no*ni + (1:(ni-1))*nw+(0:(ni-2))*nw*ni;
    H_index = Di_index + H_index(ones(nw,1),:);
    Di_index = Di_index(:);
end;

% cycle through the modes
for k=1:nl,
    D(1:nw)=Di(:,k);
    if (ni>1), H(H_index)=Di(Di_index,k); end;
    kk = find((l(k)==l)&((1:nl)<k));
    if (length(kk)==0),
        % single roots and the 1st instance of repeated roots
        if (realonly),
            num = [real(D);imag(D)];
            den = [real(H);imag(H)];
        else,
            num = D;
            den = H;
        end;
    else,
        % 2nd and subsequent instances of repeated roots
        D2 = normv(Di(:,kk)).';
        H2 = [zeros(length(kk),no) ((D2./(mpv(1,kk).'))*ones(1,ni-1)).*(mpv(2:ni,kk).')];
        if (realonly),
            num = [real([D;D2]);imag([D;D2])];
            den = [real([H;H2]);imag([H;H2])];
        else,
            num = [D;D2];
            den = [H;H2];
        end;
    end;
    % check rank
    if ~all(size(warn)),
        denrank = rank(den);
        if (denrank<min(size(den))),
            warn = sprintf(['MRMV warning:  rank deficiency (matrix is' ...
                            ' %g-by-%g, rank is %g).'],size(den),denrank);
        end;
    end;
    % compute it
    rmve = den \ num;
    rmvn=rmve(1:no); rmvn=rmvn/norm(rmvn);
    mpfn=[1; -rmve(no-1+(2:ni))]; mpfn=mpfn/norm(mpfn);
    rmv(:,k)=rmvn;
    mpv(:,k)=mpfn;
end;

% version 2, faster, but much more memory and doesn't handle repeated roots.
%lc=conj(l);
```

mrmv.m — Modified Reciprocal Modal Vector (cont.)

```
%ww=w(:,ones(1,length(l)));
%ll=l(ones(nw,1),:);
%lc=lc(ones(nw,1),:);
%Di = 1./(ww+lc*sqrt(-1)) - 1./(ww+ll*sqrt(-1));
%k=(1:nl); k=k(ones(ni-1,1),:); k=k(:)';
%DD = [Di(:,k); zeros(nw*(ni-1),nl*(ni-1))];
%DD2=zeros(nw*(ni-1),ni*nl);
%DD2(:)=DD(:); clear('DD');
%k=(1:(ni-1))';
%k2=(0:(nl-1))*ni;
%k=k(:,ones(1,nl))+k2(ones(ni-1,1),:);
%xq = resp_allw(w_index,:)
%[mq,nq] = size(xq);
%H=zeros(mq*no,nq/no);
%[ii,jj]=meshgrid(((1:no)-1)*nq*mq/no,1:mq);
%[ii,jj]=meshgrid(((1:(nq/no))-1)*mq,ii(:)+jj(:));
%H(:)=x(ii(:)+jj(:));
%clear('ii','jj','xq');
%H=[H [zeros(nw,nl*(ni-1));DD2(:,k(:)')]];
%clear('DD2');
%rmve = H \ [Di;zeros(nw*(ni-1),nl)];
%clear('H');
%rmv=rmve(1:no,:); rmv=rmv/norm(rmv);
%k=(no+(1:(ni-1)))'; k2=(0:(nl-1))*(no+nl*(ni-1));
%k=k(:,ones(1,nl))+k2(ones(ni-1,1),:);
%mpv=zeros(ni-1,nl); mpv(:)=rmve(k(:));
%mpv=[ones(1,nl);mpv]; mpv=mpv/norm(mpv);


% If the system is "square" and the sensors and actuators are collocated,
% then we could "fix" the last mode.  Note that this would be cheating,
% but it seems to produce consistently better results.  We do not do it
% here because the general system is not square and collocated.
%if (size(rmv,1)==size(rmv,2)),
%   rmv(:,no) = [-inv(rmv(1:(no-1),1:(no-1)).')*(rmv(no,1:(no-1)).');1];
%   rmv(:,no) = rmv(:,no)/norm(rmv(:,no));
%end;

% compute the rest of the outputs
if (nargout>=3),
    %ufrf=? (#rows=length(w_allw),#cols=nl*(ni+(ni>1)))
    %ufrf is the filtered (hopefully, uncoupled) frequence responses
    [um,un] = size(resp_allw);
    ufrf=sbys2stack(resp_allw,un/no)*rmv;
    ufrf=stack2sbys(ufrf,un/no);
    if (ni>1),
        k1=(1:nl:(nl*ni))';
        k2=1:nl;
        k=k1(:,ones(1,nl))+k2(ones(ni,1),:)-1;
        k=k(:)';
        k2=k2(ones(um,1),:); k2=k2(:)';
        utmp=zeros(um,nl);
        utmp(:)=sum(sbys2stack(ufrf(:,k),size(k,2)/ni)'.*mpv(:,k2))';
        ufrf=[ufrf utmp]; clear('utmp');
    end;
    if (nargout>=4),
        %robind=? (#rows=no, #cols=nl)
        %robind is the matrix of modal filter robustness indicators
        for k=1:no,
            h_norm(k,1)=norm(resp(:,k+no*(0:(ni-1))),'fro');
        end;
        Di_norm = normv(Di);
        for k=1:nl,
            robind(:,k)=abs(rmv(:,k).*h_norm)/Di_norm(1,k);
            if (ni>1), robind(:,k)=robind(:,k)*mpv(1,k); end;
        end;
        if (nargout>=5),
            condnum = normv(robind);
            if (nargout>=6),
                % compute the error and correlation info
```

mrmv.m — Modified Reciprocal Modal Vector (cont.)

```
                k = (1:nl)'; k=k(:,ones(1,ni+(ni>1))); k=k(:)';
                k2 = mpv(1,:);
                k2 = mpv ./ k2(ones(size(mpv,1),1),:);
                k2 = [k2(:).' ones(1,nl*(ni>1))];
                ufrf_tmp = ufrf(w_index,:)./k2(ones(nw,1),:);
                err_calc = zeros(nl,ni+(ni>1));
                cor_calc = zeros(nl,ni+(ni>1));
                err_calc(:)=(normv(Di(:,k)-ufrf_tmp)./Di_norm(1,k)).';
                ttmp=sum(conj(Di(:,k)).*ufrf_tmp);
                ttmp=real(ttmp.*conj(ttmp))./(Di_norm(1,k).*normv(ufrf_tmp)).^2;
                cor_calc(:) = ttmp.';
                same_index = (length(w_index)==length(w_allw));
                if (same_index),
                    same_index = all(w_index==((1:length(w_allw))'));
                end;
                if (same_index),
                    err_total = err_calc;
                    cor_total = cor_calc;
                else,
                    err_total = zeros(nl,ni+(ni>1));
                    cor_total = zeros(nl,ni+(ni>1));
                    ww=w_allw(:,ones(1,nl));
                    Di = 1./(ww+lc(ones(length(w_allw),1),:)*sqrt(-1)) ...
                       - 1./(ww+ l(ones(length(w_allw),1),:)*sqrt(-1));
                    if (out_order~=0), Di=Di.*((ww*sqrt(-1)).^out_order); end;
                    clear('ww');
                    Di_norm = normv(Di);
                    ufrf_tmp = ufrf./k2(ones(size(ufrf,1),1),:);
                    err_total(:)=(normv(Di(:,k)-ufrf_tmp)./Di_norm(1,k)).';
                    ttmp=sum(conj(Di(:,k)).*ufrf_tmp);
                    ttmp=real(ttmp.*conj(ttmp))./(Di_norm(1,k).*normv(ufrf_tmp)).^2;
                    cor_total(:) = ttmp.';
                end;
                clear('ufrf_tmp','ttmp');
            end;
        end;
    end;
end;
```

8.1.2 `mrmv_test_adapt.m` — Evaluation of MRMV

`mrmv_test_adapt` evaluates the MRMV method by testing its performance for various changes in the system characteristics. This is done, without loss of generality, on a 6 degree of freedom system.

```
% mrmv_test_adapt.m
%
% Show examples of how the output of the modal filter degrades as
% the true modeshapes change, even assuming no noise in the system
%
% Also show examples of how the modal filter still works when there
% is a complete frequency shift, or modal damping ratios change
%
% This assumes complete sensor and input knowledge

% Copyright (c)1996, Erik A. Johnson <johnsone@uiuc.edu>, 1/20/96


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initial stuff
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% printer vs. screen output:  set to '' for screen; to {'eps','ps'} for printer
%                                                   NOTE: not 'epsc' or 'psc'!!
outdev=''; %str2mat('eps','ps');

% some parameters
n=6;
w=[0;logspace(-1,1,401).'];
realrmv = 1;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ground test
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% get the system
[a,b,c,d,M,C,K,PP,l]=ndof(n,[],'displacement');
% simulate the ground test; assume it's perfect
resp = zeros(length(w),n^2);
for k=1:n, resp(:,(1:n)+(k-1)*n)=freqresp(a,b,c,d,k,sqrt(-1)*w); end;
% compute the modal filter using the "interesting" part of the freq. range
rmv = mrmv(l(1:n),w,resp,n,w>=.1&w<=3,'displacement',realrmv)
rmv_exact = PP.';
rmv_exact = rmv_exact * diag(sign(rmv_exact(1,:)./rmv(1,:)))
rmv_rel_err = (rmv-rmv_exact)./rmv_exact


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% do several tests with one element of mass matrix modified
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
massfractions = [1 1.2 1.5 2];
nmfs = length(massfractions);
ufrfs  = zeros(length(w),2*n*nmfs);
ufrfs2 = zeros(length(w),2*n*nmfs);
PPs = zeros(n,n*nmfs);
wn = zeros(n,nmfs);
for k=1:nmfs,
    currmassfrac=ones(1,n); currmassfrac(ceil(n/2))=massfractions(k);
    [a,b,c1,d1,M,C,K,PP]=ndof(n,[],'displacement',currmassfrac);
    [a,b,c2,d2,M,C,K,PP]=ndof(n,[],'acceleration',currmassfrac);
    wn(:,k) = flipud(sort(sqrt(eig(inv(M)*K))));
    PPs(:,(1:n)+(k-1)*n) = PP.';
    curresp = freqresp(a,b,[c1;c2],[d1;d2],1,sqrt(-1)*w);
    ufrfs( :,(1:n)+(k-1)*n)       = curresp(:,1:n)      * rmv;
    ufrfs2(:,(1:n)+(k-1)*n)       = curresp(:,1:n)      * PP.';
    ufrfs( :,(1:n)+(k-1+nmfs)*n) = curresp(:,n+1:2*n) * rmv;
    ufrfs2(:,(1:n)+(k-1+nmfs)*n) = curresp(:,n+1:2*n) * PP.';
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

mrmv_test_adapt.m — Evaluation of MRMV (cont.)

```
% plot the results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% scale for comparison purposes
tmp=abs([ufrfs( 2,1:nmfs*n) ufrfs( length(w),nmfs*n+1:2*nmfs*n)]);
ufrfs =ufrfs ./tmp(ones(length(w),1),:);
tmp=abs([ufrfs2(2,1:nmfs*n) ufrfs2(length(w),nmfs*n+1:2*nmfs*n)]);
ufrfs2=ufrfs2./tmp(ones(length(w),1),:);
outtypes='da';
pcolors = (1:nmfs).'*[1 1 1]/(nmfs+.05);
if (all(size(outdev))), pcolors=1-pcolors; end;
for freqnum = [1 6];
    freqshifts = (wn(freqnum,:)/wn(freqnum,massfractions==1)-1)*100;
    sprintfformat = ['%' num2str(max(diff(find([' ' ...
                    sprintf('%.1f ',freqshifts)]==' '))-1) '.1f\n'];
    legendText = sprintf(sprintfformat,freqshifts);
    legendText = strrep(strrep(strrep(legendText,' ','  '),'-','-'), ...
                        sprintf('\n'), sprintf('%% freq. shift\n'));
    legendText = str2strmat(legendText);
    legendText = str2mat(setstr([ones(size(legendText,1),1)*'\9\times' legendText]), ...
                         '\times\9{ }','\times\9estimated','\times\9actual');
    for k=1:length(outtypes),
        outtype = outtypes(k);
        % plot magnitude
            clf('reset');
            h=loglog(w,abs(ufrfs(:,(freqnum:n:n*nmfs)+(k-1)*n*nmfs)), ...
                     w,abs(ufrfs2(:,(freqnum:n:n*nmfs)+(k-1)*n*nmfs)),'--');
            % set the colors
            for kk=1:nmfs, set(h([kk kk+nmfs]),'Color',pcolors(kk,:)); end;
            set(gca,'XLim',[.1 5]);
            sxlabel('\times\12frequency [{\smaller\frac{rads}{sec}}]');
            sylabel('\times\12transfer function magnitude');
            % do the legend
            ax=axis; l=line(ax(1),ax(3),'Visible','off');
            [hleg,hlines]=slegend([.71 .25],[h(1:nmfs);l;h([1 1+nmfs])],legendText);
            set(hlines(length(hlines)-2),'Visible','off');
            set(hlines(length(hlines)+(-1:0)),'Color','w');
            % print magnitude plot
            filename = ['mrmv_test_adapt_1mass_' outtype num2str(n+1-freqnum) 'mag'];
            figure(gcf); drawnow; if (~all(size(outdev))), title(filename);pause;
            else, for od=1:size(outdev,1), outd=deblank(outdev(od,:));
            if strcmp(outd,'ps'), title(filename);drawnow; end;
            printsto(['-d' outd 'c'],[filename '.' outd]); end; end;
        % adjust phases for prettiness
            angle_ufrfs  = angle(ufrfs( :,(freqnum:n:n*nmfs)+(k-1)*n*nmfs));
            angle_ufrfs(:,2:3)=unwrap(angle_ufrfs(:,2:3));
            angle_ufrfs2 = angle(ufrfs2(:,(freqnum:n:n*nmfs)+(k-1)*n*nmfs));
            mask = angle_ufrfs(2,:)<-10*eps | angle_ufrfs(2,:)>=pi-10*eps;
            angle_ufrfs(:,mask) = angle_ufrfs(:,mask) ...
                                - ones(length(w),1)*floor(angle_ufrfs(2,mask)/pi)*pi;
            mask = angle_ufrfs2(2,:)<-10*eps | angle_ufrfs2(2,:)>=pi-10*eps;
            angle_ufrfs2(:,mask) = angle_ufrfs2(:,mask) ...
                                 - ones(length(w),1)*floor(angle_ufrfs2(2,mask)/pi)*pi;
        % plot phase
            clf('reset');
            h=semilogx(w,angle_ufrfs,w,angle_ufrfs2,'--');
            % set the colors
            for kk=1:nmfs, set(h([kk kk+nmfs]),'Color',pcolors(kk,:)); end;
            set(gca,'XLim',[.1 5]);
            sxlabel('\times\12frequency [{\smaller\frac{rads}{sec}}]');
            sylabel('\times\12transfer function phase [rads]');
            % do the legend
            ax=axis; l=line(ax(1),ax(3),'Visible','off');
            [hleg,hlines]=slegend([.33 1-.3*k],[h(1:nmfs);l;h([1 1+nmfs])],legendText);
            set(hlines(length(hlines)-2),'Visible','off');
            set(hlines(length(hlines)+(-1:0)),'Color','w');
            % print phase plot
            filename = ['mrmv_test_adapt_1mass_' outtype num2str(n+1-freqnum) 'pha'];
            figure(gcf); drawnow; if (~all(size(outdev))), title(filename);pause;
            else, for od=1:size(outdev,1), outd=deblank(outdev(od,:));
            if strcmp(outd,'ps'), title(filename);drawnow; end;
```

mrmv_test_adapt.m — Evaluation of MRMV (cont.)

```
            printsto(['-d' outd 'c'],[filename '.' outd]); end; end;
    end;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% do several tests with all elements of mass matrix modified
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
massfractions = [1 1.2 1.5 2];
nmfs = length(massfractions);
ufrfs  = zeros(length(w),2*n*nmfs);
ufrfs2 = zeros(length(w),2*n*nmfs);
PPs = zeros(n,n*nmfs);
wn = zeros(n,nmfs);
for k=1:nmfs,
    [a,b,c1,d1,M,C,K,PP]=ndof(n,[],'displacement',massfractions(k));
    [a,b,c2,d2,M,C,K,PP]=ndof(n,[],'acceleration',massfractions(k));
    wn(:,k) = flipud(sort(sqrt(eig(inv(M)*K))));
    PPs(:,(1:n)+(k-1)*n) = PP.';
    curresp = freqresp(a,b,[c1;c2],[d1;d2],1,sqrt(-1)*w);
    ufrfs( :,(1:n)+(k-1)*n)      = curresp(:,1:n)     * rmv;
    ufrfs2(:,(1:n)+(k-1)*n)      = curresp(:,1:n)     * PP.';
    ufrfs( :,(1:n)+(k-1+nmfs)*n) = curresp(:,n+1:2*n) * rmv;
    ufrfs2(:,(1:n)+(k-1+nmfs)*n) = curresp(:,n+1:2*n) * PP.';
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot the results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
freqnum = 1;
% scale for comparison purposes
tmp=abs([ufrfs( 2,1:nmfs*n) ufrfs( length(w),nmfs*n+1:2*nmfs*n)]);
ufrfs =ufrfs ./tmp(ones(length(w),1),:);
tmp=abs([ufrfs2(2,1:nmfs*n) ufrfs2(length(w),nmfs*n+1:2*nmfs*n)]);
ufrfs2=ufrfs2./tmp(ones(length(w),1),:);
outtypes='da';
pcolors = (1:nmfs).'*[1 1 1]/(nmfs+.05);
if (all(size(outdev))), pcolors=1-pcolors; end;
freqshifts = (wn(freqnum,:)/wn(freqnum,massfractions==1)-1)*100;
sprintfformat = ['%' num2str(max(diff(find([' ' sprintf('%.1f ', ...
                                freqshifts)]==' ')))-1) '.1f\n'];
legendText = sprintf(sprintfformat,freqshifts);
legendText = strrep(strrep(strrep(legendText,' ','  '),'-','-'), ...
                    sprintf('\n'),sprintf('%% freq. shift\n'));
legendText = str2strmat(legendText);
legendText = str2mat(setstr([ones(size(legendText,1),1)*'\9\times' legendText]), ...
                     '\times\9( )','\times\9estimated','\times\9actual');
for k=1:length(outtypes),
    outtype = outtypes(k);
    % plot magnitude
        clf('reset');
        h=loglog(w,abs(ufrfs(:,(freqnum:n:n*nmfs)+(k-1)*n*nmfs)), ...
                 w,abs(ufrfs2(:,(freqnum:n:n*nmfs)+(k-1)*n*nmfs)),'--');
        % set the colors
        for kk=1:nmfs, set(h([kk kk+nmfs]),'Color',pcolors(kk,:)); end;
        set(gca,'XLim',[.1 5]);
        sxlabel('\times\12frequency [{\smaller\frac{rads}{sec}}]');
        sylabel('\times\12transfer function magnitude');
        % do the legend
        ax=axis; l=line(ax(1),ax(3),'Visible','off');
        [hleg,hlines]=slegend([.33 .35],[h(1:nmfs);l;h([1 1+nmfs])],legendText);
        set(hlines(length(hlines)-2),'Visible','off');
        set(hlines(length(hlines)+(-1:0)),'Color','w');
        % print magnitude plot
        filename = ['mrmv_test_adapt_allm_' outtype num2str(n+1-freqnum) 'mag'];
        figure(gcf); drawnow; if (~all(size(outdev))), title(filename);pause;
        else, for od=1:size(outdev,1), outd=deblank(outdev(od,:));
        if strcmp(outd,'ps'), title(filename);drawnow; end;
        printsto(['-d' outd 'c'],[filename '.' outd]); end; end;
    % adjust phases for prettiness
        angle_ufrfs  = angle(ufrfs( :,(freqnum:n:n*nmfs)+(k-1)*n*nmfs));
```

mrmv_test_adapt.m — Evaluation of MRMV (cont.)

```
        angle_ufrfs(:,2:3)=unwrap(angle_ufrfs(:,2:3));
        angle_ufrfs2 = angle(ufrfs2(:,(freqnum:n:n*nmfs)+(k-1)*n*nmfs));
        mask = angle_ufrfs(2,:)<-10*eps | angle_ufrfs(2,:)>=pi-10*eps;
        angle_ufrfs(:,mask) = angle_ufrfs(:,mask) ...
                            - ones(length(w),1)*floor(angle_ufrfs(2,mask)/pi)*pi;
        mask = angle_ufrfs2(2,:)<-10*eps | angle_ufrfs2(2,:)>=pi-10*eps;
        angle_ufrfs2(:,mask) = angle_ufrfs2(:,mask) ...
                            - ones(length(w),1)*floor(angle_ufrfs2(2,mask)/pi)*pi;
    % plot phase
        clf('reset');
        h=semilogx(w,angle_ufrfs,w,angle_ufrfs2,'--');
        % set the colors
        for kk=1:nmfs, set(h([kk kk+nmfs]),'Color',pcolors(kk,:)); end;
        set(gca,'XLim',[.1 5]);
        sxlabel('\times\12frequency [{\smaller\frac{rads}{sec}}]');
        sylabel('\times\12transfer function phase [rads]');
        % do the legend
        ax=axis; l=line(ax(1),ax(3),'Visible','off');
        [hleg,hlines]=slegend([.33 .5],[h(1:nmfs);l;h([1 1+nmfs])],legendText);
        set(hlines(length(hlines)-2),'Visible','off');
        set(hlines(length(hlines)+(-1:0)),'Color','w');
        % print phase plot
        filename = ['mrmv_test_adapt_allm_' outtype num2str(n+1-freqnum) 'pha'];
        figure(gcf); drawnow; if (~all(size(outdev))), title(filename);pause;
        else, for od=1:size(outdev,1), outd=deblank(outdev(od,:));
        if strcmp(outd,'ps'), title(filename);drawnow; end;
        printsto(['-d' outd 'c'],[filename '.' outd]); end; end;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% do several tests with modal damping ratio modified
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
massfractions = [1 1.2 1.5 2];
nmfs = length(massfractions);
ufrfs  = zeros(length(w),2*n*nmfs);
ufrfs2 = zeros(length(w),2*n*nmfs);
PPs = zeros(n,n*nmfs);
wn = zeros(n,nmfs);
for k=1:nmfs,
    [a,b,c1,d1,M,C,K,PP]=ndof(n,[],'displacement');
    [a,b,c2,d2,M,C,K,PP]=ndof(n,[],'acceleration');
    Z = diag([massfractions(k);ones(n-1,1)])*(inv(PP)'*C*inv(PP));
    C = PP'*Z*PP;
    a(n+1:2*n,n+1:2*n) = -inv(M)*C;
    c2(:,n+1:2*n) = -inv(M)*C;
    wn(:,k) = diag(Z)./flipud(sort(sqrt(eig(inv(M)*K))))/2;
    PPs(:,(1:n)+(k-1)*n) = PP.';
    curresp = freqresp(a,b,[c1;c2],[d1;d2],1,sqrt(-1)*w);
    ufrfs( :,(1:n)+(k-1)*n)      = curresp(:,1:n)     * rmv;
    ufrfs2(:,(1:n)+(k-1)*n)      = curresp(:,1:n)     * PP.';
    ufrfs( :,(1:n)+(k-1+nmfs)*n) = curresp(:,n+1:2*n) * rmv;
    ufrfs2(:,(1:n)+(k-1+nmfs)*n) = curresp(:,n+1:2*n) * PP.';
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot the results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
freqnum = 1;
% scale for comparison purposes
tmp=abs([ufrfs( 2,1:nmfs*n) ufrfs( length(w),nmfs*n+1:2*nmfs*n)]);
ufrfs =ufrfs ./tmp(ones(length(w),1),:);
tmp=abs([ufrfs2(2,1:nmfs*n) ufrfs2(length(w),nmfs*n+1:2*nmfs*n)]);
ufrfs2=ufrfs2./tmp(ones(length(w),1),:);
outtypes='da';
pcolors = (1:nmfs).'*[1 1 1]/(nmfs+.05);
if (all(size(outdev))), pcolors=1-pcolors; end;
freqshifts = (wn(freqnum,:)/wn(freqnum,massfractions==1)-1)*100;
sprintfformat = ['%' num2str(max(diff(find([' ' sprintf('%.1f ', ...
                                freqshifts)]==' '))-1) '.1f\n'];
legendText = sprintf(sprintfformat,freqshifts);
```

mrmv_test_adapt.m — Evaluation of MRMV (cont.)

```
legendText = strrep(strrep(strrep(legendText,' ','  '),'-','-'), ...
                    sprintf('\n'),sprintf('%% damping increase\n'));
legendText = str2strmat(legendText);
legendText = str2mat(setstr([ones(size(legendText,1),1)*'\9\times' legendText]), ...
                     '\times\9{ }','\times\9estimated','\times\9actual');
for k=1:length(outtypes),
    outtype = outtypes(k);
    % plot magnitude
        clf('reset');
        h=loglog(w,abs(ufrfs(:,(freqnum:n:n*nmfs)+(k-1)*n*nmfs)), ...
                 w,abs(ufrfs2(:,(freqnum:n:n*nmfs)+(k-1)*n*nmfs)),'--');
        % set the colors
        for kk=1:nmfs, set(h([kk kk+nmfs]),'Color',pcolors(kk,:)); end;
        set(gca,'XLim',[.1 5]);
        sxlabel('\times\12frequency [{\smaller\frac{rads}{sec}}]');
        sylabel('\times\12transfer function magnitude');
        % do the legend
        ax=axis; l=line(ax(1),ax(3),'Visible','off');
        [hleg,hlines]=slegend([.31 .75],[h(1:nmfs);l;h([1 1+nmfs])],legendText);
        set(hlines(length(hlines)-2),'Visible','off');
        set(hlines(length(hlines)+(-1:0)),'Color','w');
        % print magnitude plot
        filename = ['mrmv_test_adapt_damp_' outtype num2str(n+1-freqnum) 'mag'];
        figure(gcf); drawnow; if (~all(size(outdev))), title(filename);pause;
        else, for od=1:size(outdev,1), outd=deblank(outdev(od,:));
        if strcmp(outd,'ps'), title(filename);drawnow; end;
        printsto(['-d' outd 'c'],[filename '.' outd]); end; end;
    % adjust phases for prettiness
        angle_ufrfs  = angle(ufrfs( :,(freqnum:n:n*nmfs)+(k-1)*n*nmfs));
        angle_ufrfs(:,2:3)=unwrap(angle_ufrfs(:,2:3));
        angle_ufrfs2 = angle(ufrfs2(:,(freqnum:n:n*nmfs)+(k-1)*n*nmfs));
        mask = angle_ufrfs(2,:)<-10*eps | angle_ufrfs(2,:)>=pi-10*eps;
        angle_ufrfs(:,mask) = angle_ufrfs(:,mask) ...
                              - ones(length(w),1)*floor(angle_ufrfs(2,mask)/pi)*pi;
        mask = angle_ufrfs2(2,:)<-10*eps | angle_ufrfs2(2,:)>=pi-10*eps;
        angle_ufrfs2(:,mask) = angle_ufrfs2(:,mask) ...
                               - ones(length(w),1)*floor(angle_ufrfs2(2,mask)/pi)*pi;
    % plot phase
        clf('reset');
        h=semilogx(w,angle_ufrfs,w,angle_ufrfs2,'--');
        % set the colors
        for kk=1:nmfs, set(h([kk kk+nmfs]),'Color',pcolors(kk,:)); end;
        set(gca,'XLim',[.1 5]);
        sxlabel('\times\12frequency [{\smaller\frac{rads}{sec}}]');
        sylabel('\times\12transfer function phase [rads]');
        % do the legend
        ax=axis; l=line(ax(1),ax(3),'Visible','off');
        [hleg,hlines]=slegend([.4 .45],[h(1:nmfs);l;h([1 1+nmfs])],legendText);
        set(hlines(length(hlines)-2),'Visible','off');
        set(hlines(length(hlines)+(-1:0)),'Color','w');
        % print magnitude plot
        % print phase plot
        filename = ['mrmv_test_adapt_damp_' outtype num2str(n+1-freqnum) 'pha'];
        figure(gcf); drawnow; if (~all(size(outdev))), title(filename);pause;
        else, for od=1:size(outdev,1), outd=deblank(outdev(od,:));
        if strcmp(outd,'ps'), title(filename);drawnow; end;
        printsto(['-d' outd 'c'],[filename '.' outd]); end; end;
end;
```

8.1.3 mrmvtool.m — Graphical User Interface for mrmv function

mrmvtool is a graphical user interface (GUI) front-end to mrmv. It is run just by executing it with no arguments. Computing reciprocal modal vectors, filtering data, choosing spectral lines to use in computing the vectors, and plotting filtered and unfiltered data can be done with mrmvtool.

```
function [h,h2,h3,h4,h5,h6]=mrmvtool(action,flag,val,arg4,arg5,arg6,arg7)
% MRMVTOOL  Graphical User Interface for MRMV (modified reciprocal modal filter)
%
%   MRMVTOOL provides a graphical user interface to MRMV (modified
%            reciprocal modal filter).
%
%   Call MRMVTOOL with no arguments to begin.
%
%   Call MRMVTOOL DEMO for a demonstration (requires mrmvtool_demo.m).
%
%   Requires Matlab 4.2 and the UITools toolbox from The MathWorks.

% Copyright (c)1996, Erik A. Johnson <johnsone@uiuc.edu>, 3/30/96

if (nargin==1), if strcmp(lower(action),'demo'), mrmvtool_demo; return; end; end;

% input arguments
if (nargin<1), action=[]; end;
if (nargin<2), flag=[]; end;
if (nargin<3), val=[]; end;
if isempty(action), action='initialize'; end;

% some constants
mrmv_name = 'MRMV Tool';
mrmv_options = 'MRMV Tool Options';
bgcolor = [1 1 1]*.75;
bgcolor_edit = [1 1 1]*.9;
bgcolor_frame = [1 1 1]*.625;
toolbarheight = 0.07;
buttontextsize = 10;
plotlocs = [.06 .07 .92 .55];
extravertspace = .02;
indiceslineserase = 'background'; %'normal'
debugging = 0;

% check for existing tool
mrmv_fig = findobj(get(0,'Children'),'flat','Name',mrmv_name);

% check possible actions
if (isempty(mrmv_fig) | strcmp(action,'initialize')),
    % take care of existing figure
    if ~isempty(mrmv_fig),
        if (debugging),
            close('all');
        else,
            figure(mrmv_fig);
            axes(findobj(mrmv_fig,'Type','axes','Tag','zoom'));
            return;
        end;
    end;

    % initialize figure
    mrmv_fig = figure('Visible','off','Color',bgcolor, ...
                      'Name',mrmv_name,'Interruptible','yes', ...
                      'Nextplot','new','NumberTitle','off', ...
                      'KeyPressFcn','mrmvtool(''keycall'')', ...
                      'WindowButtonMotionFcn','mrmvtool(''moved'')', ...
                      'Units','pixels');
    set(mrmv_fig,'Pointer','watch'); drawnow;
    if (debugging & (exist('paperaxes')>1)), paperaxes; end;
    colormap(gray(17));
    fig_ar=get(mrmv_fig,'Position');
    if (fig_ar(3)==0), fig_ar=1; else, fig_ar=fig_ar(4)/fig_ar(3); end;
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
% some callback functions
computecb = ['mrmvtool(''compute''' ...
             ',eval(mrmvtool(''getstr'',''unfiltered''),''''''err'''''')' ...
             ',eval(mrmvtool(''getstr'',''freqs''     ),''''''err'''''')' ...
             ',eval(mrmvtool(''getstr'',''indices''   ),''''''err'''''')' ...
             ',eval(mrmvtool(''getstr'',''nrefs''     ),''''''err'''''')' ...
             ',eval(mrmvtool(''getstr'',''poles''     ),''''''err'''''')' ...
             ')'];
plotcb = ['mrmvtool(''plot''' ...
          ',eval(mrmvtool(''getstr'',''unfiltered''),''''''err'''''')' ...
          ',eval(mrmvtool(''getstr'',''freqs''     ),''''''err'''''')' ...
          ',eval(mrmvtool(''getstr'',''indices''   ),''''''err'''''')' ...
          ')'];

% initialize menus
mrmv_menu = uimenu(mrmv_fig,'Label','MRMVTool');
uimenu(mrmv_menu,'Label','Plot','Interruptible','yes','CallBack',plotcb);
uimenu(mrmv_menu,'Label','Compute RMVs','Interruptible','yes','CallBack',computecb);
uimenu(mrmv_menu,'Label','Zoom In','Interruptible','yes', ...
       'Separator','on','CallBack','mrmvtool(''zoom'',''in'')');
uimenu(mrmv_menu,'Label','Zoom Out','Interruptible','yes', ...
       'CallBack','mrmvtool(''zoom'',''out'')');
uimenu(mrmv_menu,'Label','Restore Limits','Interruptible','yes', ...
       'CallBack','mrmvtool(''zoom'',''restore'')');
uimenu(mrmv_menu,'Label','Options...','Interruptible','yes', ...
       'Separator','on','CallBack','mrmvtool(''options'')');
uimenu(mrmv_menu,'Label','Exit MRMV','Interruptible','yes', ...
       'Separator','on','CallBack','mrmvtool(''exit'')');
drawnow;

% initialize axes
ax1 = axes('Position',plotlocs,'XColor','k','YColor','k','Color','w', ...
           'DrawMode','fast','box','on','XScale','log','YScale','log', ...
           'Interruptible','yes','ButtonDownFcn',['mrmvtool(''choose'',' ...
           'eval(mrmvtool(''getstr'',''indices''),''''''err'''''')'], ...
           'Tag','ax1');
xlabel('Frequency [rads/sec]');
hl=[get(ax1,'XLabel');get(ax1,'YLabel')];
set(hl,'FontWeight','bold','FontSize',round(get(hl(1),'FontSize')*.8));

% bar above plot
p = [plotlocs(1) plotlocs(2)+plotlocs(4)+extravertspace 0 0] ...
  + toolbarheight*[0 0 fig_ar 1];
ptop = p(2)+p(4);
btngroup(mrmv_fig,'GroupID','zoom','ButtonID','zoom','PressType','toggle', ...
         'IconFunctions','mrmvtool(''btndraw'',''zoom'')', ...
         'CallBacks','mrmvtool(''btndo'',''zoom'')', ...
         'Orientation','horizontal','Position',p);
p(1) = p(1) + p(3) + extravertspace*fig_ar;
p(3) = plotlocs(1)+plotlocs(3) - p(1);
uicontrol(mrmv_fig,'Style','frame','Units','normalized','Position',p, ...
          'BackgroundColor',bgcolor_frame);
fractvertspace = .2*toolbarheight;
horizspace = fractvertspace*fig_ar;
p = p + horizspace*[1 0 -2 0] + fractvertspace*[0 1 0 -2];
buttonheight = p(4);
uicontrol(mrmv_fig,'Style','pushbutton','Units','normalized', ...
          'Position',[p(1)+p(3)*0.00 p(2) p(3)*.1 p(4)],'String','Plot', ...
          'CallBack',plotcb,'Tag','plot');
xlabels = str2mat('Frequency [rads/sec]','Frequency [rads/sec]', ...
                  'Mode','Mode','Mode','Mode','Sensor Number');
uicontrol(mrmv_fig,'Style','popupmenu','Units','normalized', ...
          'HorizontalAlignment','left', ...
          'Position',[p(1)+p(3)*0.1 p(2) p(3)*.325 p(4)],'Tag','popup', ...
          'String',['Unfiltered Data|Filtered Data|UFRF Error (indexed)|' ...
                    'UFRF Error (all)|UFRF Correlation (indexed)|' ...
                    'UFRF Correlation (all)|RMV Matrix'], ...
          'UserData',[[1 1;1 1;0 0;0 0;0 0;0 0;0 0] xlabels]);
uicontrol(mrmv_fig,'Style','pushbutton','Units','normalized', ...
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
          'Position',[p(1)+p(3)*0.475 p(2) p(3)*.2 p(4)], ...
          'String','to workspace as','CallBack','eval(mrmvtool(''toworkspace''))');
uicontrol(mrmv_fig,'Style','edit','Units','normalized', ...
          'HorizontalAlignment','left', ...
          'Position',[p(1)+p(3)*0.675 p(2) p(3)*.325 p(4)],'String','', ...
          'BackgroundColor',bgcolor_edit,'Tag','toworkspace');
% top stuff
p = [plotlocs(1) ptop+extravertspace plotlocs(3) 1-extravertspace];
p(4) = p(4) - p(2);
uicontrol(mrmv_fig,'Style','frame','Units','normalized','Position',p, ...
          'BackgroundColor',bgcolor_frame);
nxy = [2 3];
vs = (p(4)-nxy(2)*buttonheight) / (nxy(2)+1);
p = p + vs*[fig_ar 1 -fig_ar -1];
p(3:4) = p(3:4) ./ nxy;
dxy = p(3:4);
p(3:4) = p(3:4) - vs*[fig_ar 1];
textfactors = [.16 .7];
uicontrol(mrmv_fig,'Style','text','BackgroundColor',bgcolor_frame, ...
          'String','Unfiltered Data:','Units','normalized', ...
          'Position',p.*[1 1 .32 textfactors(2)] ...
                    +[dxy.*[0 2]+p(3:4).*[0 textfactors(1)] 0 0], ...
          'HorizontalAlignment','right');
uicontrol(mrmv_fig,'Style','edit','BackgroundColor',bgcolor_edit, ...
          'HorizontalAlignment','left', ...
          'String','[]','Units','normalized','Tag','unfiltered', ...
          'Position',p.*[1 1 2/3 1]+[dxy.*[0 2]+p(3:4).*[1/3 0] 0 0]);
uicontrol(mrmv_fig,'Style','text','BackgroundColor',bgcolor_frame, ...
          'String','Frequencies:','Units','normalized', ...
          'Position',p.*[1 1 .32 textfactors(2)] ...
                    +[dxy.*[0 1]+p(3:4).*[0 textfactors(1)] 0 0], ...
          'HorizontalAlignment','right');
uicontrol(mrmv_fig,'Style','edit','BackgroundColor',bgcolor_edit, ...
          'HorizontalAlignment','left', ...
          'String','[]','Units','normalized','Tag','freqs', ...
          'Position',p.*[1 1 2/3 1]+[dxy.*[0 1]+p(3:4).*[1/3 0] 0 0]);
uicontrol(mrmv_fig,'Style','text','BackgroundColor',bgcolor_frame, ...
          'String','# of Refs.:','Units','normalized', ...
          'Position',p.*[1 1 .32 textfactors(2)] ...
                    +[dxy.*[0 0]+p(3:4).*[0 textfactors(1)] 0 0], ...
          'HorizontalAlignment','right');
uicontrol(mrmv_fig,'Style','edit','BackgroundColor',bgcolor_edit, ...
          'HorizontalAlignment','left', ...
          'String','1','Units','normalized','Tag','nrefs', ...
          'Position',p.*[1 1 2/3 1]+[dxy.*[0 0]+p(3:4).*[1/3 0] 0 0]);
uicontrol(mrmv_fig,'Style','text','BackgroundColor',bgcolor_frame, ...
          'String','Freq. Indices:','Units','normalized', ...
          'Position',p.*[1 1 .32 textfactors(2)] ...
                    +[dxy.*[1 2]+p(3:4).*[0 textfactors(1)] 0 0], ...
          'HorizontalAlignment','right');
uicontrol(mrmv_fig,'Style','edit','BackgroundColor',bgcolor_edit, ...
          'HorizontalAlignment','left', ...
          'String',''''all''','Units','normalized','Tag','indices', ...
          'Position',p.*[1 1 2/3 1]+[dxy.*[1 2]+p(3:4).*[1/3 0] 0 0]);
uicontrol(mrmv_fig,'Style','text','BackgroundColor',bgcolor_frame, ...
          'String','Poles:','Units','normalized', ...
          'Position',p.*[1 1 .32 textfactors(2)] ...
                    +[dxy.*[1 1]+p(3:4).*[0 textfactors(1)] 0 0], ...
          'HorizontalAlignment','right');
uicontrol(mrmv_fig,'Style','edit','BackgroundColor',bgcolor_edit, ...
          'HorizontalAlignment','left', ...
          'String','[]','Units','normalized','Tag','poles', ...
          'Position',p.*[1 1 2/3 1]+[dxy.*[1 1]+p(3:4).*[1/3 0] 0 0]);
p(1) = p(1) + dxy(1);
button_labels = str2mat('Compute RMVs','Options...','Exit');
button_cbacks = str2mat(computecb ...
                       ,'mrmvtool(''options'')' ...
                       ,'mrmvtool(''exit'')' ...
                      );
button_tags   = str2mat('compute','','');
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
    nx = size(button_labels,1);
    button_widths = zeros(nx,1);
    % define the font and size
    % these are hard coded on most platforms in
    %     Version 4, but may vary from site to site.
    c = computer;
    if all(c(1:2)=='MA'),      fontsize=10; fontname='Geneva';
    elseif all(c(1:2)=='PC'), fontsize=[]; fontname='';
    else,                      fontsize=[]; fontname='';
    end;
    % set them up
    oldax = get(mrmv_fig,'CurrentAxes');
    set(mrmv_fig,'CurrentAxes',ax1);
    axlims = axis;
    t_junk = text(axlims(1),axlims(3),deblank(button_labels(1,:)));
    set(t_junk,'Units','pixels');
    if ~isempty(fontname), set(t_junk,'FontName',fontname); end;
    if ~isempty(fontsize), set(t_junk,'FontSize',fontsize); end;
    ext = get(t_junk,'extent');
    button_widths(1) = ext(3);
    for kk=2:nx,
        set(t_junk,'String',deblank(button_labels(kk,:)));
        ext = get(t_junk,'extent');
        button_widths(kk) = ext(3);
    end;
    delete(t_junk);
    set(mrmv_fig,'CurrentAxes',oldax);
    figpos = get(mrmv_fig,'Position');
    extrawidth = figpos(3) * (p(3)-vs*fig_ar*(nx-1)) - sum(button_widths);
    button_widths = button_widths + extrawidth / nx;
    button_widths = (p(3)-vs*fig_ar*(nx-1))*button_widths/sum(button_widths);
    for kk=1:nx,
        p(3) = button_widths(kk);
        uicontrol(mrmv_fig,'Style','pushbutton', ...
                  'String',deblank(button_labels(kk,:)), ...
                  'CallBack',deblank(button_cbacks(kk,:)), ...
                  'Tag',deblank(button_tags(kk,:)), ...
                  'Units','normalized','Position',p);
        p(1) = p(1) + p(3) + vs*fig_ar;
    end;
    % set up default properties
    mrmvtool('set','oldunfiltered','');
    mrmvtool('set','RMVMatrix','');
    mrmvtool('set','MPVMatrix','');
    mrmvtool('set','FilteredData','');
    mrmvtool('set','UFRFErrorindexed','');
    mrmvtool('set','UFRFErrorall','');
    mrmvtool('set','UFRFCorrelationindexed','');
    mrmvtool('set','UFRFCorrelationall','');
    % anything else?
    if (debugging),
        % some dummy data
        set(findobj(mrmv_fig,'Style','edit','Tag','unfiltered'), ...
            'String','[1 5;2 6;3 7;4 8;4 9;3 8;2 6;1 8]');
        set(findobj(mrmv_fig,'Style','edit','Tag','freqs'),'String','1:8');
    end;
    % make it visible
    figure(mrmv_fig);
    ax=findobj(mrmv_fig,'Type','axes','Tag','zoom');
    if ~isempty(ax), set(mrmv_fig,'CurrentAxes',ax); end;
    whitebg(mrmv_fig,[1 1 1]*.75);
    set(mrmv_fig,'Pointer','arrow','Visible','on');

elseif strcmp(action,'exit'),
    allfigs = findobj(get(0,'Children'),'flat','Type','figure');
    myfigs = mrmvtool('get','subfigs');
    for k=myfigs(:).', if any(k==allfigs), delete(k); end; end;
    close(mrmv_fig);
    if (debugging), mrmvtool; end;
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
elseif strcmp(action,'getstr'),
    if isempty(flag),
        h = [];
    else,
        h = get(findobj(mrmv_fig,'Tag',flag),'String');
    end;

elseif strcmp(action,'setstr'),
    if isstr(val),
        str = val;
    elseif isempty(val),
        str = '[]';
    elseif (all(val(:)==round(val(:))) & all(val(:)>=0) & any(size(val)>2)),
        trans=0; if (size(val,2)==1), trans=1; val=val.'; end;
        val = [val.';-2*ones(1,size(val,1))];
        val = val(:).';
        ii = ([diff(val) 0]==1);
        ii = ii & ([1 diff(ii)]==0);
        val(ii) = -3 + 0*val(ii);
        ii = ii & ([1 diff(ii)]==0);
        val(ii) = [];
        str = sprintf('%.0f ',val);
        str = strrep(strrep(str,' -3 ',':'),' -2 ',';');
        str(length(str)) = [];
        str = ['[' str ']'];
        if (trans), str=[str '.''']; end;
    else,
        str = mat2str(val);
    end;
    set(findobj(mrmv_fig,'Tag',flag),'String',str);

elseif strcmp(action,'getfig'),
    h = mrmv_fig;

elseif strcmp(action,'popup'),
    popup = findobj(mrmv_fig,'Tag','popup');
    if isempty(popup),
        error('MRMVTOOL cannot find its popup menu.');
    end;
    v      = get(popup,'Value');
    minval = get(popup,'Min');
    maxval = get(popup,'Max');
    if (v<minval | v>maxval),
        error('MRMVTOOL seems to have a corrupted popup menu.');
    end;
    if isempty(flag),
        h = get(popup,'Value');
    else,
        if isstr(flag),
            if (size(flag,2)==1), flag=flag.'; end;
            if (size(flag,1)~=1),
                error(['MRMVTOOL requires a string row vector ' ...
                       'when setting the popup menu value.']);
            end;
            s = lower(str2mat(flag,get(popup,'String')));
            s(s==0) = s(s==0)+' ';
            ii = find(all((ones(size(s,1),1)*s(1,:)==s).'));
            if (length(ii)==1),
                for k=1:size(s,1),
                    s2 = s(k,:);
                    s2(s2==' '|s2=='('|s2==')') = [];
                    s(k,:) = setstr([s2 ' '*ones(1,size(s,2)-length(s2))]);
                end;
                ii = find(all((ones(size(s,1),1)*s(1,:)==s).'));
            end;
            ii(1) = [];
            if all(size(ii)),
                flag = min(ii)-1;
            else,
                error(['MRMVTOOL does not recognize ''' ...
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
                    flag ''' as a valid popup menu value.']);
            end;
        elseif (flag<minval | flag>maxval),
            error(['MRMVTOOL requires the popup value to be in [' ...
                   num2str(minval) ',' num2str(maxval) '].']);
        end;
        set(popup,'Value',flag);
    end;

elseif strcmp(action,'toworkspace'),
    str = mrmvtool('getstr','toworkspace');
    if ~isempty(str), str=[str '=']; end;
    disp([str 'mrmvtool(''getplotdata'',...);']);
    h = [str 'mrmvtool(''getplotdata''' ...
         ',eval(mrmvtool(''getstr'',''unfiltered''),''''''err'''''')' ...
         ',eval(mrmvtool(''getstr'',''freqs''     ),''''''err'''''')' ...
         ',eval(mrmvtool(''getstr'',''indices''   ),''''''err'''''')' ...
         ',1);'];

elseif strcmp(action,'checkrefilter'),
    % get the data (flag=unfiltered,val=freqs,arg4=indices,arg5=errordialog)
    h=0;
    oldflag  = mrmvtool('get','oldunfiltered');
    if any(size(oldflag)~=size(flag)),
        neednew = 1;
    else,
        neednew = any(flag(:)~=oldflag(:));
    end;
    if (neednew), % need to recompute
        % get the filters
        rmv = mrmvtool('get','RMVMatrix');
        mpv = mrmvtool('get','MPVMatrix');
        [no,nl] = size(rmv);
        % check the data first
        mrmvtool('checkparse','unfiltered',flag);
        if any([no nl]==0),
            errmsg = 'MRMVTOOL cannot filter the data until the RMVs are computed.';
            if (arg5),
                mrmvtool('error','error',errmsg);
            else,
                error(errmsg);
            end;
            h=1;
        elseif (rem(size(flag,2),no)~=0),
            errmsg = ['MRMVTOOL requires that the # of columns of the Unfiltered ' ...
                      'Data be a ' sprintf('\n') 'multiple of the # of outputs ' ...
                      'when the RMVs were computed.'];
            if (arg5),
                mrmvtool('error','error',errmsg);
            else,
                error(errmsg);
            end;
            h=1;
        else,
            % recompute the uncoupled frfs
            ni = size(flag,2)/no;
            [um,un] = size(flag);
            ufrf=stack2sbys(sbys2stack(flag,un/no)*rmv,un/no);
            mpvWeighted = [];
            if (ni>1 & size(mpv,1)==ni),
                k1=(1:nl:(nl*ni))';
                k2=1:nl;
                k=k1(:,ones(1,nl))+k2(ones(ni,1),:)-1;
                k=k(:)';
                k2=k2(ones(um,1),:); k2=k2(:)';
                utmp=zeros(um,nl);
                utmp(:)=sum(sbys2stack(ufrf(:,k),size(k,2)/ni)'.*mpv(:,k2))';
                mpvWeighted = size(ufrf,2)+(1:nl);
                ufrf=[ufrf utmp]; clear('utmp');
            end;
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
            % save them again
            mrmvtool('set','oldunfiltered',flag);
            mrmvtool('set','FilteredData',ufrf);
            mrmvtool('set','MPVWeightedFilteredData',mpvWeighted);
        end;
    end;

elseif strcmp(action,'getplotdata'),
    % get the data (flag=unfiltered,val=freqs,arg4=indices)
    dialogonerror = (nargout>=6) | (nargin>=5);
    popup = findobj(mrmv_fig,'Tag','popup');
    if isempty(popup),
        error('MRMVTOOL cannot find its popup menu.');
    end;
    v = get(popup,'Value');
    if ((v<get(popup,'Min')) | (v>get(popup,'Max'))),
        error('MRMVTOOL seems to have a corrupted popup menu.');
    end;
    logs = get(popup,'UserData');
    logs = logs(v,:);
    h6=0;
    s = get(popup,'String');
    s = deblank(s(v,:));
    ss = s;
    s((s==' ')|(s=='(')|(s==')')) = [];
    if (v==1), % unfiltered
        mrmvtool('checkparse','unfiltered',flag);
    else,
        if (v==2), h6=mrmvtool('checkrefilter',flag,val,arg4,dialogonerror); end; %filtered
        flag = mrmvtool('get',s);
    end;
    if (~h6 & isstr(flag)),
        errmsg = ['MRMVTOOL has not yet computed ''' ss '''.'];
        if (dialogonerror),
            mrmvtool('error','error',errmsg);
        else,
            error(errmsg);
        end;
        h6=1;
    end;
    % compute outputs
    h=flag;
    if (h6),
        h2 = [];
    elseif (v==1|v==2), %frequency-based data
        mrmvtool('checkparse','freqs',val);
        if (min(size(val))>1),
            errmsg = 'MRMVTOOL requires the frequencies be a vector, not a matrix.'
            if (dialogonerror),
                mrmvtool('error','error',errmsg);
            else,
                error(errmsg);
            end;
            h = [];
            h6 = 1;
        end;
        h2 = val(:);
    else,
        h2 = (1:size(flag,1)).';
        % {err,cor}_{index,total} are (nl)-by-(ni+(ni>1))
        % rmv is (no)-by-(nl)
        % mpv is (ni)-by-(nl)
    end;
    h3=logs;
    h4=v;
    h5=popup;

elseif strcmp(action,'plot'),
    if (nargin==1),
        h = get(findobj(mrmv_fig,'Tag','plot'),'CallBack');
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
    return;
end;
% check if currently log scale
ax = findobj(mrmv_fig,'Tag','ax1');
if isempty(ax), error('MRMVTOOL cannot find its axes.'); end;
waslog = [strcmp(get(ax,'XScale'),'log') strcmp(get(ax,'YScale'),'log')];
% check old log/linear scale
oldv = mrmvtool('get','oldv');
if ~isempty(oldv),
    alllogs = get(popup,'UserData');
    if (oldv>=1&oldv<=size(alllogs,1)),
        alllogs(oldv,1:2) = waslog;
    end;
    set(popup,'UserData',alllogs);
end;
% get the data (flag=unfiltered,val=freqs,arg4=indices)
[pd,x,logs,v,popup,err] = mrmvtool('getplotdata',flag,val,arg4);
if (err), return; end;
% check it
if isempty(pd),
    mrmvtool('error','error', ...
             'MRMVTOOL cannot plot what is empty or not yet computed.');
    return;
elseif isempty(x),
    x = (1:size(pd,1)).';
elseif (size(x,1)~=size(pd,1)),
    mrmvtool('error','error', ...
             ['MRMVTOOL requires that the Unfiltered Data and the' ...
              sprintf('\n') 'Frequencies have the same # of rows.']);
    return;
end;
% handle absolute values
warn = '';
if (v==1|v==2),
    pd=abs(pd); %we are only doing magnitude plots here
elseif (v==7 & ~isreal(pd)),
    pd=abs(pd);
    if mrmvtool('get','showwarnings'),
        warn = 'MRMVTOOL warning:  the RMVs are complex; plotting abs(RMVs).';
    end;
end;
% do we have a MPV-weighted filtered frf?
ii = [];
if (v==2),
    ii = mrmvtool('get','MPVWeightedFilteredData');
end;
% plot it
oldfig = gcf;
figure(mrmv_fig);
oldax = gca;
axes(ax);
delete(get(ax,'Children'));
l = line(x,pd,'ButtonDownFcn',get(ax,'ButtonDownFcn'));
if all(size(ii)), set(l(ii),'Color',mrmvtool('get','MPVWeightedColor')); end;
set(get(ax,'XLabel'),'String',setstr(logs(3:length(logs))));
if (v>2), set(ax,'XTick',x); else, set(ax,'XTickMode','auto'); end;
mrmvtool('set','lastplotted',v);
% handle log scales
waslog = [strcmp(get(ax,'XScale'),'log') strcmp(get(ax,'YScale'),'log')];
if (waslog(1)~=logs(1)),
    if (logs(1)), s='log'; else, s='linear'; end;
    set(ax,'XScale',s);
end;
if (waslog(2)~=logs(2)),
    if (logs(2)), s='log'; else, s='linear'; end;
    set(ax,'YScale',s);
end;
% handle limits
if (v>2),
    xlims = [1-min(1,(max(x)-1)/10) max(x)+min(1,(max(x)-1)/10)];
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
    else,
        xlims = [min(x) max(x)];
    end;
    set(ax,'XLim',xlims,'YLimMode','auto');
    set(get(ax,'ZLabel'),'UserData',[]);
    % clean up
    axes(oldax);
    figure(oldfig);
    ax=findobj(mrmv_fig,'Type','axes','Tag','zoom');
    if ~isempty(ax), set(mrmv_fig,'CurrentAxes',ax); end;
    if all(size(warn)), mrmvtool('error','warning',warn); end;

elseif strcmp(action,'centerloc'),
    if (all(size(flag)) & ~isstr(flag)),
        flag = flag(1);
        if     (flag==0), flag='mouse';
        elseif (flag==1), flag='mouse';
        elseif (flag==2), flag='mrmvtool';
        else,             flag='screen';
        end;
    end;
    flag = lower(flag);
    if     strcmp(flag,'screen'),   pos=get(0,'ScreenSize');pos=pos(1:2)+pos(3:4)/2;
    elseif strcmp(flag,'mrmvtool'), pos=get(mrmv_fig,'Position');pos=pos(1:2)+pos(3:4)/2;
    else,                           pos=get(0,'PointerLocation');
    end;
    h = pos;

elseif strcmp(action,'options') & isempty(flag),
    % set up and display the options dialog
    oldfig = gcf;
    oldpointer = get(oldfig,'Pointer');
    if ~strcmp(oldpointer,'watch'), set(oldfig,'Pointer','watch'); end;
    dlg = findobj('Type','figure','Name',mrmv_options);
    if ~isempty(dlg),
        figure(dlg);
    else,
        labels = str2mat( ...
                 str2mat('Data Output Type','text','0','') ...
                ,str2mat('displacement','radiobutton','1','disp') ...
                ,str2mat('velocity','radiobutton','1','vel') ...
                ,str2mat('acceleration','radiobutton','1','accel') ...
                ,str2mat('','','0','') ...
                ,str2mat('Restrict RMVs to Real Numbers','checkbox','-1','realrmv') ...
                ,str2mat('Display Warnings','checkbox','-1','warnbox') ...
                );
        [m,n]   = size(labels);
        m       = m/4;
        tags    = labels(4:4:4*m,:);
        indents = setstr([ones(m,1)*' ' labels(3:4:4*m,:)]).';
        indents = eval(['[' indents(:).' '].''']);
        styles  = labels(2:4:4*m,:);
        labels  = labels(1:4:4*m,:);
        nchars  = n - sum(cumprod(fliplr(labels).'==' '|fliplr(labels).'==0)).';
        % some size parameters
        mCharacterWidth = 7;
        Voff = 5;
        layout
        mPushbuttonWidth  = mStdButtonWidth;
        mPushbuttonHeight = mStdButtonHeight;
        mIndentWidth      = mFrameToText;
        mCheckBoxWidth    = mFrameToText;
        mLineHeight       = max(mLineHeight,mPushbuttonHeight);
        mOKString         = mOkButtonString;
        mCancelString     = mCancelButtonString;
        % compute size of dialog box
        FigWH = [(max(nchars*mCharacterWidth ...
                    +(indents>0).*indents*mIndentWidth ...
                    -(indents<0).*indents*mCheckBoxWidth)) ...
                 (m*(mLineHeight+Voff)-Voff+mPushbuttonHeight+ ...
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
                4*mFrameToText+3*mEdgeToFrame)];
        FigWH(1) = max(FigWH(1),2*mPushbuttonWidth+2*mEdgeToFrame+3*mFrameToText);
        % compute location
        pos = mrmvtool('centerloc');
        pos = [pos-FigWH/2 FigWH];
        % create the figure
        DefUIBgColor = get(0,'DefaultUIControlBackgroundColor');
        dlg = figure('NumberTitle','off','Name',mrmv_options,'Units','pixels', ...
                     'Position',pos,'NextPlot','new','MenuBar','none', ...
                     'Color',DefUIBgColor,'Visible','off');
        mrmvtool('set','subfigs',[mrmvtool('get','subfigs');dlg]);
        % make the 2 frame uicontrols
        UIPos1 = mEdgeToFrame*[1 1 -2 0] + [0 0 FigWH(1) mLineHeight+2*mFrameToText];
        set(uicontrol(dlg,'Style','frame','Position',UIPos1),'Units','normalized');
        UIPos2 = [UIPos1(1:3)+[0 UIPos1(4)+mEdgeToFrame 0] ...
                  m*(mLineHeight+Voff)-Voff+2*mFrameToText];
        set(uicontrol(dlg,'Style','frame','Position',UIPos2),'Units','normalized');
        % make the OK and Cancel buttons
        Hspace = (FigWH(1) - 2*mPushbuttonWidth) / 3;
        set(uicontrol(dlg,'Style','pushbutton','String',mOKString, ...
                      'Callback','mrmvtool(''options'',''OK'')', ...
                      'Position',[UIPos1(1:2)+[Hspace mFrameToText] ...
                                  mPushbuttonWidth mPushbuttonHeight]),'Units','normalized');
        set(uicontrol(dlg,'Style','pushbutton','String',mCancelString, ...
                      'Callback','mrmvtool(''options'',''Cancel'')', ...
                      'Position',[UIPos1(1:2)+[2*Hspace+mPushbuttonWidth mFrameToText] ...
                                  mPushbuttonWidth mPushbuttonHeight]),'Units','normalized');
        % set up the rest
        UIPos2 = [UIPos2(1:2)+[mFrameToText m*(mLineHeight+Voff)+Voff/2+mFrameToText] ...
                  UIPos2(3)-2*mFrameToText mLineHeight];
        for kk=1:m,
            if ~isempty(deblank(labels(kk,:))),
                indent = max(0,indents(kk))*mIndentWidth;
                UIPos1 = UIPos2 + [indent -kk*(mLineHeight+Voff) -indent 0];
                tag = deblank(tags(kk,:));
                if ~isempty(tag),
                    cback = ['mrmvtool(''options'',''' tag ''')'];
                else,
                    cback = '';
                end;
                set(uicontrol(dlg,'Style',deblank(styles(kk,:)), ...
                              'String',deblank(labels(kk,:)), ...
                              'Position',UIPos1,'Tag',tag,'Callback',cback, ...
                              'HorizontalAlignment','left'),'Units','normalized');
            end;
        end;
    end;
    % set values
    mrmvtool('options','realrmv',mrmvtool('get','realonly'));
    mrmvtool('options','warnbox',mrmvtool('get','showwarnings'));
    outtype = mrmvtool('get','outtype');
    mrmvtool('options','disp', outtype==0);
    mrmvtool('options','vel',  outtype==1);
    mrmvtool('options','accel',outtype==2);
    % make it visible
    if ~strcmp(oldpointer,'watch'), set(oldfig,'Pointer',oldpointer); end;
    set(dlg,'Visible','on');

elseif strcmp(action,'options'),
    % handle options dialog items
    dlg = findobj('Type','figure','Name',mrmv_options);
    obj = findobj(dlg,'Tag',flag);
    if isempty(dlg),
        mrmvtool('options');
    elseif strcmp(flag,'OK'),
        mrmvtool('set','realonly',mrmvtool('options','realrmv','get')~=0);
        mrmvtool('set','showwarnings',mrmvtool('options','warnbox','get')~=0);
        if     (mrmvtool('options','disp', 'get')~=0), outtype=0;
        elseif (mrmvtool('options','vel',  'get')~=0), outtype=1;
        elseif (mrmvtool('options','accel','get')~=0), outtype=2;
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
        else,                                                outtype=[];
        end;
        mrmvtool('set','outtype',outtype);
        set(dlg,'Visible','off');
        figure(mrmv_fig);
    elseif strcmp(flag,'Cancel'),
        set(dlg,'Visible','off');
        figure(mrmv_fig);
    elseif strcmp(flag,'realrmv') | strcmp(flag,'warnbox'),
        if isempty(obj),
            error('MRMVTOOL has a corrupted options dialog.');
        elseif strcmp(val,'get'),
            h = get(obj,'Value');
        else,
            if ~isempty(val), set(obj,'Value',val); end;
        end;
    elseif strcmp(flag,'disp') | strcmp(flag,'vel') | strcmp(flag,'accel'),
        if isempty(obj),
            error('MRMVTOOL has a corrupted options dialog.');
        elseif strcmp(val,'get'),
            h = get(obj,'Value');
        elseif isempty(val),
            val = get(obj,'Value')~=0;
            if (~val),
                set(obj,'Value',~val);
            else,
                s = str2mat(flag,'disp','vel','accel');
                ii = find(all((ones(size(s,1),1)*s(1,:)==s).'));
                s(ii,:) = [];
                mrmvtool('options',deblank(s(1,:)),~val);
                mrmvtool('options',deblank(s(2,:)),~val);
            end;
        else,
            set(obj,'Value',val);
        end;
    else,
        mrmvtool('error','badargs',['MRMVTOOL does not recognize ''' ...
                                setstr(flag(:).') ''' as an options flag.']);
    end;

elseif strcmp(action,'compute'),
    if (nargin==1),
        h = get(findobj(mrmv_fig,'Tag','compute'),'CallBack');
        return;
    elseif (nargin<6),
        mrmvtool('error','badargs', ...
                 'MRMVTOOL requires 6 input arguments to compute the RMVs.');
        return;
    end;
    % check data
    mrmvtool('checkparse','unfiltered',flag);
    mrmvtool('checkparse','freqs'     ,val);
    mrmvtool('checkparse','indices'   ,arg4);
    mrmvtool('checkparse','nrefs'     ,arg5);
    mrmvtool('checkparse','poles'     ,arg6);
    if isstr(arg4), if strcmp(arg4,'none'), arg4=[]; else, arg4=1:size(flag,1); end; end;
    % check consistency
    if (size(val,1)~=size(flag,1)),
        mrmvtool('error','error', ...
                 'MRMVTOOL requires the Data and Frequency lengths be the same.');
        return;
    elseif any(arg4<1 | arg4>length(val)),
        mrmvtool('error','error', ...
                 'MRMVTOOL requires that the frequency indices be valid.');
        return;
    elseif (arg5~=round(arg5) | rem(size(flag,2),arg5)~=0),
        mrmvtool('error','error',['MRMVTOOL requires that the # of columns ' ...
                                  'of the Unfiltered Data be a multiple of #ofRefs.']);
    elseif isempty(arg6),
        mrmvtool('error','error', ...
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
                'MRMVTOOL needs one or more poles from which to compute RMVs.');
        return;
    end;
    % do the computing
    set(mrmv_fig,'Pointer','watch');
    val      = val(:);
    arg4     = arg4(:);
    outtype  = mrmvtool('get','outtype');
    realonly = mrmvtool('get','realonly');
    [rmv,mpv,ufrfs,robind,condnum,err_calc,err_total,cor_calc,cor_total,warn] ...
                        = mrmv(arg6,val,flag,arg5,arg4,outtype,realonly,1);
    set(mrmv_fig,'Pointer','arrow');
    if isstr(rmv),
        mrmvtool('error','error',rmv);
    else,
        mrmvtool('set','oldunfiltered',flag);
        mrmvtool('set','RMVMatrix',rmv);
        mrmvtool('set','MPVMatrix',mpv);
        mrmvtool('set','FilteredData',ufrfs);
        mrmvtool('set','UFRFErrorindexed',err_calc);
        mrmvtool('set','UFRFErrorall',err_total);
        mrmvtool('set','UFRFCorrelationindexed',cor_calc);
        mrmvtool('set','UFRFCorrelationall',cor_total);
        mrmvtool('set','MPVWeightedFilteredData',prod(size(mpv))+1:size(ufrfs,2));
        if (~isempty(warn) & mrmvtool('get','showwarnings')),
            mrmvtool('error','warning',warn);
        end;
    end;

elseif strcmp(action,'checkparse'),
    if     strcmp(flag,'unfiltered'), str = 'Unfiltered Data';
    elseif strcmp(flag,'freqs'     ), str = 'frequencies';
    elseif strcmp(flag,'indices'   ), str = 'frequency indices';
    elseif strcmp(flag,'nrefs'     ), str = 'numbers-of-references';
    elseif strcmp(flag,'poles'     ), str = 'poles';
    else, mrmvtool('error','badargs',['MRMVTOOL cannot parse a string of type `'' ...
                                      flag(:).' ''.']);
        return;
    end;
    if isstr(val),
        if strcmp(val,'err'),
            mrmvtool('error','error',['MRMVTOOL is unable to parse the ' ...
                                      str ' string.' sprintf('\n') lasterr]);
        elseif ~(strcmp(flag,'indices') & (strcmp(val,'all')|strcmp(val,'none'))),
            mrmvtool('error','error',['MRMVTOOL cannot use `'' val(:).' ...
                                      ''' as ' str '.']);
        end;
    end;

elseif strcmp(action,'choose'),
    % check if we have frequency-based data
    v = mrmvtool('get','lastplotted');
    if (v>2),
        mrmvtool('error','warning',['MRMVTOOL cannot choose spectral ' ...
                 'lines unless Unfiltered or Filtered Data is plotted.']);
        return;
    end;
    if ~isempty(flag),
        if isstr(flag),
            mrmvtool('checkparse','indices',flag);
            if strcmp(flag,'none'), flag=[]; end;
        else,
            flag = flag(:);
        end;
    end;
    % get the axes
    oldfig = gcf;
    figure(mrmv_fig);
    oldax = gca;
    ax = findobj(mrmv_fig,'Tag','ax1');
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
axes(ax);
% check if we're in bounds
if isstr(val) | any(size(val)~=[1 2]),
    pt = get(ax,'CurrentPoint');
    pt = pt(1,1:2);
else,
    pt = val;
end;
curlim = [get(ax,'XLim');get(ax,'YLim')].';
if (all(pt>=curlim(1,:)) & all(pt<=curlim(2,:))),
    if (~btnstate(mrmv_fig,'zoom','zoom')),
        popup = findobj(mrmv_fig,'Tag','popup');
        if isempty(popup),
            axes(oldax);
            figure(oldfig);
            error('MRMVTOOL cannot find its popup menu.');
        end;
        v = get(popup,'Value');
        if any(v==[1 2 4 6]),
            l = findobj(ax,'Type','line','LineStyle','o','Tag','indiceslines');
            if (length(l)~=2), delete(l); l=[]; end;
            if isempty(l),
                % turn choosing on
                % get the data points
                kids = findobj(ax,'Type','line');
                if ~isempty(kids),
                    x = get(kids(1),'XData'); x=x(:);
                    y1 = zeros(length(x),length(kids));
                    for k=1:length(kids),
                        yy = get(kids(k),'YData');
                        if (length(yy) ~= length(x)),
                            axes(oldax);
                            figure(oldfig);
                            mrmvtool('error','error',['MRMVTOOL found data' ...
                                    ' of varying lengths in plot window.']);
                            return;
                        end;
                        y1(:,k) = yy.';
                    end;
                    % check indices
                    n = length(x);
                    if strcmp(flag,'all'),
                        flag = (1:n).';
                    elseif any( (flag>n) | (flag<1) | (flag~=round(flag)) ),
                        flag = round(flag);
                        flag((flag>n)|(flag<1)) = [];
                        flag0 = flag;
                        if (length(flag)==n),
                            if all(flag(:).'==[1:n]),
                                flag0 = 'all';
                            end;
                        end;
                        if isstr(flag0), flag0=['''' flag0(:).' '''']; end;
                        mrmvtool('setstr','indices',flag0);
                    end;
                    % divy them up
                    ii = zeros(n,1);
                    ii(flag) = ones(length(flag),1);
                    x1 = x( ii,ones(1,size(y1,2)));
                    x0 = x(~ii,ones(1,size(y1,2)));
                    y0 = y1(~ii,:);
                    y1 = y1(ii,:);
                    x0=x0(:).'; x1=x1(:).'; y0=y0(:).'; y1=y1(:).';
                    if isempty(x1),
                        l(1) = line(NaN,NaN,'Color','r','LineStyle','o', ...
                                'Tag','indiceslines');
                        set(l(1),'XData',[],'YData',[]);
                    else,
                        l(1) = line(x1,y1,'Color','r','LineStyle','o', ...
                                'Tag','indiceslines');
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
            end;
            set(l(1),'UserData',x);
            if isempty(x0),
                l(2) = line(NaN,NaN,'Visible','off','LineStyle','o', ...
                            'Tag','indiceslines');
                set(l(2),'XData',[],'YData',[]);
            else,
                l(2) = line(x0,y0,'Visible','off','LineStyle','o', ...
                            'Tag','indiceslines');
            end;
            set(l,'ButtonDownFcn',get(ax,'ButtonDownFcn'), ...
                  'EraseMode',indiceslineserase);
        end;
    else,
        % get the data
        l = [findobj(l,'flat','Visible','on');findobj(l,'flat','Visible','off')];
        x = get(l(1),'UserData');
        x1 = get(l(1),'XData');
        x0 = get(l(2),'XData');
        y1 = get(l(1),'YData');
        y0 = get(l(2),'YData');
        n = length(x);
        % check the indices
        if (isstr(flag) & strcmp(flag,'all')),
            flag = (1:n).';
        elseif any( (flag>n) | (flag<1) | (flag~=round(flag)) ),
            flag = round(flag(:));
            flag((flag>n)|(flag<1)) = [];
        end;
        flag = sort(flag);
        if ~isempty(flag), flag([diff(flag);1]==0)=[]; end;
        if (length(x1)*n/(length(x1)+length(x0)) ~= length(flag)),
            % indices has been changed ... update picture to reflect it
            x0=[x0 x1]; x0=x0(:);
            y0=[y0 y1]; y0=y0(:);
            [x0,ii] = sort(x0); y0=y0(ii);
            x0=reshape(x0,length(x0)/n,n).'; y0=reshape(y0,length(y0)/n,n).';
            [junk,ii] = sort(x');
            x0(ii,:)=x0; y0(ii,:)=y0;
            x1=x0(flag,:); x0(flag,:)=[]; y1=y0(flag,:); y0(flag,:)=[];
            x0=x0(:).'; x1=x1(:).'; y0=y0(:).'; y1=y1(:).';
        end;
        % compute distances
        if strcmp(get(ax,'XScale'),'log'),
            x1d = log10(x1/pt(1))/log10(curlim(2,1)/curlim(1,1));
            x0d = log10(x0/pt(1))/log10(curlim(2,1)/curlim(1,1));
        else,
            x1d = (x1-pt(1))/(curlim(2,1)-curlim(1,1));
            x0d = (x0-pt(1))/(curlim(2,1)-curlim(1,1));
        end;
        if strcmp(get(ax,'YScale'),'log'),
            y1d = log10(y1/pt(2))/log10(curlim(2,2)/curlim(1,2));
            y0d = log10(y0/pt(2))/log10(curlim(2,2)/curlim(1,2));
        else,
            y1d = (y1-pt(2))/(curlim(2,2)-curlim(1,2));
            y0d = (y0-pt(2))/(curlim(2,2)-curlim(1,2));
        end;
        ii=find(isnan(x1d)); if (~isempty(ii)),x1d(ii)=inf*ones(length(ii),1);end;
        ii=find(isnan(x0d)); if (~isempty(ii)),x0d(ii)=inf*ones(length(ii),1);end;
        ii=find(isnan(y1d)); if (~isempty(ii)),y1d(ii)=inf*ones(length(ii),1);end;
        ii=find(isnan(y0d)); if (~isempty(ii)),y0d(ii)=inf*ones(length(ii),1);end;
        oldu = get(ax,'Units');
        oldp = get(ax,'Position');
        set(ax,'Units','pixels');
        p = get(ax,'Position');
        set(ax,'Units',oldu,'Position',oldp);
        x1d = (x1d*p(3)).^2 + (y1d*p(4)).^2;
        x0d = (x0d*p(3)).^2 + (y0d*p(4)).^2;
        % determine minumum
        [junk,ii1] = min(x1d);
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
                    [junk,ii0] = min(x0d);
                    ii = 1;
                    if (~isempty(ii1) | ~isempty(ii0)),
                        if isempty(ii1),
                            ii = 0;
                        elseif ~isempty(ii0),
                            ii = (x1d(ii1) <= x0d(ii0));
                        end;
                        % adjust data
                        if (ii),
                            ii = find(x1==x1(ii1));
                            flag(find(flag==find(x==x1(ii(1))))) = [];
                            x0 = [x0 x1(ii)];
                            y0 = [y0 y1(ii)];
                            x1(ii) = [];
                            y1(ii) = [];
                        else,
                            ii = find(x0==x0(ii0));
                            flag = sort([flag;find(x==x0(ii(1)))]);
                            flag([diff(flag);1]==0) = [];
                            x1 = [x1 x0(ii)];
                            y1 = [y1 y0(ii)];
                            x0(ii) = [];
                            y0(ii) = [];
                        end;
                        % replace it
                        set(l(1),'XData',x1,'YData',y1);
                        set(l(2),'XData',x0,'YData',y0);
                        if (length(flag)==n),
                            if all(flag(:).'==[1:n]),
                                flag = 'all';
                            end;
                        end;
                        if isstr(flag), flag=['''' flag(:).' '''']; end;
                        mrmvtool('setstr','indices',flag);
                    end;
                end;
            end;
        end;
    elseif (pt(2)<=curlim(2,2)),
        axlimdlg('Axes Limits Dialog',[1 1]);
        oldfig = gcf;
    end;
    set(mrmv_fig,'CurrentAxes',oldax);
    figure(oldfig);
    ax=findobj(mrmv_fig,'Type','axes','Tag','zoom');
    if ~isempty(ax), set(mrmv_fig,'CurrentAxes',ax); end;

elseif strcmp(action,'btndraw'),
    if strcmp(flag,'zoom'),
        if (0), % faster text version, but not as nice
            h=text(.5,.5,'zoom','HorizontalAlignment','center','FontSize',buttontextsize);
        else, % cool icon version
            x1 = [0.0917 0.1250 0.1583 0.1917 0.2250 0.2917 0.3583 0.4250 0.4583, ...
                  0.4917 0.5250 0.5250 0.5583 0.5917 0.6250 0.6583 0.6917 0.7250, ...
                  0.7583 0.7917 0.8250 0.8917 0.9250 0.9250 0.8917 0.8583 0.8250, ...
                  0.7917 0.7583 0.7250 0.6917 0.6583 0.6250 0.5583 0.5250 0.5583, ...
                  0.5917 0.5917 0.5917 0.5917 0.5583 0.5250 0.4917 0.4583 0.4250, ...
                  0.3583 0.2917 0.2250 0.1917 0.1583 0.1250 0.0917 0.0583 0.0583, ...
                  0.0583 0.0583 0.0917 0.1250 0.1250 0.1250 0.1250 0.1583 0.1917, ...
                  0.2250 0.2917 0.3583 0.4250 0.4583 0.4917 0.5250 0.5250 0.5250, ...
                  0.5250 0.4917 0.4583 0.4250 0.3583 0.2917 0.2250 0.1917 0.1583, ...
                  0.1583 0.1250];
            y1 = [0.5417 0.5083 0.4750 0.4417 0.4083 0.4083 0.4083 0.4083 0.4417, ...
                  0.4750 0.4417 0.3750 0.3417 0.3083 0.2750 0.2417 0.2083 0.1750, ...
                  0.1417 0.1083 0.0750 0.0750 0.1083 0.1750 0.2083 0.2417 0.2750, ...
                  0.3083 0.3417 0.3750 0.4083 0.4417 0.4750 0.4750 0.5083 0.5417, ...
                  0.5750 0.6417 0.7083 0.7750 0.8083 0.8417 0.8750 0.9083 0.9417, ...
                  0.9417 0.9417 0.9417 0.9083 0.8750 0.8417 0.8083 0.7750 0.7083, ...
                  0.6417 0.5750 0.5417 0.5750 0.6417 0.7083 0.7750 0.8083 0.8417, ...
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
                0.8750 0.8750 0.8750 0.8750 0.8417 0.8083 0.7750 0.7083 0.6417, ...
                0.5750 0.5417 0.5083 0.4750 0.4750 0.4750 0.4750 0.5083 0.5417, ...
                0.5417 0.5750];
            x2 = [0.2917 0.3583 0.3917 0.3917 0.3583 0.2917 0.2583 0.2583 0.2917];
            y2 = [0.6083 0.6083 0.6417 0.7083 0.7417 0.7417 0.7083 0.6417 0.6083];
            h = [patch('XData',x1,'YData',y1,'ZData',ones(size(x1)), ...
                    'EdgeColor','none','FaceColor','k'); ...
                 patch('XData',x2,'YData',y2,'ZData',ones(size(x2)), ...
                    'EdgeColor','none','FaceColor','k')];
        end;
    else,
        error(['MRMVTOOL got an unknown button ''' flag ''' to draw.']);
    end;

elseif strcmp(action,'btndo'),
    if strcmp(flag,'zoom'),
        inzoom = btnstate(mrmv_fig,'zoom','zoom');
        ax = findobj(mrmv_fig,'Tag','ax1');
        if (inzoom),
            % turning zoom on
            set(mrmv_fig,'WindowButtonDownFcn','mrmvtool(''zoom'')', ...
                        'WindowButtonUpFcn','1;','ButtonDownFcn','');
            for k=ax(:).',
                set(k,'UserData',get(k,'ButtonDownFcn'),'ButtonDownFcn','');
                set(findobj(get(k,'Children'),'Type','line'),'ButtonDownFcn','');
            end;
        else,
            % turning zoom off
            set(mrmv_fig,'WindowButtonDownFcn','','WindowButtonUpFcn','');
            for k=ax(:).',
                bdf = get(k,'UserData');
                set(k,'ButtonDownFcn',bdf,'UserData','');
                set(findobj(get(k,'Children'),'Type','line'),'ButtonDownFcn',bdf);
            end;
        end;
    else,
        error(['MRMVTOOL got an unknown button ''' flag ''' to do.']);
    end;

elseif strcmp(action,'zoom'),
    % find the axes
    if (gcf ~= mrmv_fig), return; end;
    oldax = gca;
    if isempty(flag),
        ax = mrmvtool('findax');
    else,
        ax = findobj(mrmv_fig,'Tag','ax1');
    end;
    if ~isempty(ax),
        axes(ax);
        % get the first point
        pts = get(ax,'CurrentPoint');
        pts = [pts(1,1:2);pts(1,1:2)];
        % do the zoom
        sel = get(1,'SelectionType');
        zlab = get(ax,'ZLabel');
        ud = get(zlab,'UserData');
        if (strcmp(sel,'open') | strcmp(flag,'restore')),
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % There is still a problem with this.  For some reason, the 'open' %
            % selection type (double-clicks) isn't ever occurring.  I don't    %
            % know why.  Seems to work in other windows, but not here.         %
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % zoom back to original limits
            if ~isempty(ud),
                if any(size(ud)~=[1 4]),
                    disp('MRMVTOOL was unable to zoom out because the ''UserData''');
                    disp('of the ''ZLabel'' of the axes has been corrupted.');
                else,
                    axis(ud);
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
                set(zlab,'UserData',[]);
            end;
        end;
    else,
        if isempty(flag),
            selnorm = strcmp(sel,'normal');
        else,
            selnorm = strcmp(flag,'in');
        end;
        % store the limits for later restoration
        if isempty(ud), set(zlab,'UserData',[get(ax,'XLim') get(ax,'YLim')]); end;
        % get the second point if needed
        if (selnorm & isempty(flag)),
            % rbbox requires the figure units be pixels
            oldu = get(mrmv_fig,'Units');
            if (~strcmp(oldu,'pixels')),
                oldp = get(mrmv_fig,'Position');
                set(mrmv_fig,'Units','pixels');
            end;
            % do the rubber-band box
            rbbox([get(mrmv_fig,'CurrentPoint') 0 0],get(mrmv_fig,'CurrentPoint'));
            % restore figure units and position
            if (~strcmp(oldu,'pixels')),
                set(mrmv_fig,'Units',oldu);
                set(mrmv_fig,'Position',oldp);
            end;
            % get the second point
            pt2 = get(ax,'CurrentPoint');
            pts(2,:) = pt2(1,1:2);
        end;
        % adjust limits for log scaling
        curlim = [get(ax,'XLim');get(ax,'YLim')].';
        logs = [strcmp(get(ax,'XScale'),'log') strcmp(get(ax,'YScale'),'log')];
        logs = [logs;logs];
        if (any(logs(:))),
            curlim(logs) = log10(curlim(logs));
            pts(   logs) = log10(pts(   logs));
        end;
        % for menus, make it the middle
        if ~isempty(flag),
            pts = [1;1]/2*sum(curlim);
        end;
        % check proximity of points in units of pixels
        if (selnorm & isempty(flag)),
            oldu = get(ax,'Units');
            oldp = get(ax,'Position');
            set(ax,'Units','pixels');
            p = get(ax,'Position');
            set(ax,'Units',oldu,'Position',oldp);
            if (sqrt(sum((diff(pts)./diff(curlim).*p(3:4)).^2)) <= 3),
                pts(2,:) = pts(1,:);
            end;
        end;
        % determine new limits
        if all(diff(pts)==0),
            factor = 2;
            if (selnorm), factor=1/factor; end;
            curlim = factor/2 * diff(curlim);
            curlim = pts + [-curlim;curlim];
        else,
            curlim = [min(pts);max(pts)];
        end;
        % readjust for log scales
        if (any(logs(:))), curlim(logs)=10.^curlim(logs); end;
        % set the new limits
        set(ax,'XLim',curlim(:,1).','YLim',curlim(:,2).');
    end;
end;
axes(oldax);
ax=findobj(mrmv_fig,'Type','axes','Tag','zoom');
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
    if ~isempty(ax), set(mrmv_fig,'CurrentAxes',ax); end;

elseif strcmp(action,'keycall'),
    char = get(mrmv_fig,'CurrentCharacter');
    if abs(char)==127 | abs(char)==8,
        mrmvtool('clear');
    elseif abs(char)==12,
        refresh(mrmv_fig);
    end;

elseif strcmp(action,'findax'),
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % I have no idea why the drawnow/drawnow('discard') is necessary,  %
    % but if they are not present and if the window is larger than a   %
    % certain size then we get a redraw every time this is called.     %
    % This may only occur on certain platforms, but in any case, the   %
    % drawnow pair should take care of the problem for the time being. %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    drawnow;
    pt = get(mrmv_fig,'CurrentPoint');
    ax = findobj(mrmv_fig,'Tag','ax1');
    oldu = get(ax,'Units');
    oldp = get(ax,'Position');
    set(ax,'Units','pixels');
    p = get(ax,'Position');
    set(ax,'Units',oldu);
    set(ax,'Position',oldp);
    drawnow('discard');
    if ((pt(1)>=p(1)) & (pt(2)>=p(2)) & (pt(1)<=p(1)+p(3)) & (pt(2)<=p(2)+p(4))),
        h = ax;
    else,
        h = [];
    end;

elseif strcmp(action,'moved'),
    % check where we are and adjust pointer accordingly
    if (mrmv_fig == gcf),
        inzoom = btnstate(mrmv_fig,'zoom','zoom');
        curptr = get(mrmv_fig,'Pointer');
        ax = mrmvtool('findax');
        if (inzoom & ~isempty(ax)),
            newptr = 'cross';
        elseif ((~isempty(ax)) & (~isempty(findobj(ax,'Type','line', ...
                                'LineStyle','o','Tag','indiceslines')))),
            newptr = 'crosshair';
        else,
            newptr = 'arrow';
        end;
        if ~strcmp(curptr,newptr),
            set(mrmv_fig,'Pointer',newptr);
        end;
        ax=findobj(mrmv_fig,'Type','axes','Tag','zoom');
        if ~isempty(ax), set(mrmv_fig,'CurrentAxes',ax); end;
    end;

elseif strcmp(action,'clear'),
    if (debugging),
        disp('MRMVTOOL does nothing with a ''clear'' action.');
    end;

elseif strcmp(action,'get'),
    % this is coded like this such that if getuprop() or
    % setuprop() are unavailable a work-around could be coded
    h = getuprop(mrmv_fig,flag);
    if isempty(h),
        if     strcmp(flag,'realonly'),     h=0;
        elseif strcmp(flag,'outtype'),      h=0;
        elseif strcmp(flag,'showwarnings'), h=1;
        elseif strcmp(flag,'MPVWeightedColor'),
            h=get(findobj(mrmv_fig,'Tag','ax1'),'DefaultLineColor');
```

mrmvtool.m — Graphical User Interface for mrmv function (cont.)

```
        end;
    end;

elseif strcmp(action,'set'),
    % coded for easy work-around if getuprop() or setuprop() are unavailable
    setuprop(mrmv_fig,flag,val);

elseif strcmp(action,'error'),
    if strcmp(flag,'badargs'), % bad inputs somehow
        error(val);
    elseif strcmp(flag,'error') | strcmp(flag,'warning'),
        oldfig = gcf;
        oldpointer = get(oldfig,'Pointer');
        if ~strcmp(oldpointer,'watch'), set(oldfig,'Pointer','watch'); end;
        % get the dialog
        dialogname=flag; dialogname(1)=upper(dialogname(1));
        dialogname = ['MRMVTool ' dialogname];
        h = findobj('Type','figure','Name',dialogname);
        hnew = ~all(size(h));
        centerpos = mrmvtool('centerloc',1);
        if (hnew),
            h = dialog('Style',flag,'TextString',val,'Name',dialogname,'Replace','on');
            set(h,'Units','pixels');
        elseif (length(h)>1),
            delete(h(2:length(h)));
            h = h(1);
        end;
        % center it
        pos = get(h,'Position');
        pos(1:2) = centerpos - pos(3:4)/2;
        set(h,'Position',pos);
        % if impossible, center over mrmv_fig
        pos = get(h,'Position');
        if any(centerpos<pos(1:2) | centerpos>pos(1:2)+pos(3:4)),
            centerpos=mrmvtool('centerloc',2);
            pos(1:2) = centerpos - pos(3:4)/2;
            set(h,'Position',pos);
        end;
        pos = get(h,'Position');
        if any(centerpos<pos(1:2) | centerpos>pos(1:2)+pos(3:4)),
            centerpos=mrmvtool('centerloc',3);
            pos(1:2) = centerpos - pos(3:4)/2;
            set(h,'Position',pos);
        end;
        % set the text and properties
        errortexttag = 'errortext';
        if (hnew),
            closefunc = ['set(gcf,''Visible'',''off'');set(findobj(gcf,' ...
                         '''Tag'',''' errortexttag '''),''String'','''');'];
            set(h,'KeyPressFcn',['if (abs(get(gcf,''CurrentChar''))==13' ...
                '|abs(get(gcf,''CurrentChar''))==3), ' closefunc ' end']);
            set(findobj(h,'CallBack','delete(gcf)'),'CallBack',closefunc);
            set(findobj(h,'String',val),'Tag',errortexttag);
            mrmvtool('set','subfigs',[mrmvtool('get','subfigs');h]);
        else,
            set(findobj(h,'Tag',errortexttag),'String',val);
        end;
        if ~strcmp(oldpointer,'watch'), set(oldfig,'Pointer',oldpointer); end;
        figure(h);
        set(h,'Visible','on');
        drawnow;
    else, % anything else
        error(val);
    end;

else,
    error(['MRMVTOOL got an unknown action: ''' action '''.']);

end;
```

8.1.4 mrmvtool_demo.m — Demo Script for MRMVTool

mrmvtool_demo runs a simple demonstration of some of the capabilities of the mrmvtool GUI.

```
% mrmvtool_demo   Demonstrate the MRMVTOOL GUI.

% Copyright (c)1996, Erik A. Johnson <johnsone@uiuc.edu>, 5/30/96

clc
echo on

% This script demonstrates some of the abilities of the MRMVTool GUI
% front-end to MRMV.

% First set up some data.  A simple 4-degree of freedom system, with
% 2 inputs and 3 outputs (not collocated).

n=4;   nrefs=2;   no=3;

[a,b,c,d,M,C,K,PP,l,W] = ndof(n);
poles=l(1:n);
[a,b,c,d] = ssselect(a,b,c,d,size(b,2)+(1-nrefs:0),1:no);

Wrange = (max(W)/min(W))^(1/3);
frqs=logspace(log10(min(W)/Wrange),log10(max(W)*Wrange),401).';

frfs = zeros(length(frqs),nrefs*no);
for k=1:nrefs, frfs(:,(1:no)+(k-1)*no)=freqresp(a,b,c,d,k,sqrt(-1)*frqs); end;



pause  % Press any key to continue after pauses.

clc

% Now we start up the MRMVTOOL graphical user interface

mrmvtool;



pause  % Press any key.

clc

% The editable text strings must be set to the data computed above.
% They will generally be typed in by hand, but from an m-file we can do:

mrmvtool('setstr','unfiltered','frfs');
mrmvtool('setstr','freqs','frqs');
mrmvtool('setstr','nrefs','nrefs');
mrmvtool('setstr','indices','''all''');
mrmvtool('setstr','poles','poles');

% (The GUI functions below will all be done using a command-line, but
% the user actions to do the same thing will be given in parenthesis.)



pause  % Press any key.

clc

% The Unfiltered Data can be plotted
% (select 'Unfiltered Data' from the popup menu and hit the 'Plot' button)

mrmvtool('popup','Unfiltered Data');
eval(mrmvtool('plot'));
```

mrmvtool_demo.m — Demo Script for MRMVTool (cont.)

```
pause  % Press any key.

clc

% The axes limits can be changed by clicking anywhere below or left of the
% main plot axes.  This will bring up the 'Axes Limits Dialog'.

mrmvtool('choose',eval(mrmvtool('getstr','indices'),'''err'''),[-inf -inf]);
axlimdlg = gcf;


pause  % Press any key.


% The axes limits can be changed and the dialog closed when done.

if any(findobj==axlimdlg), delete(axlimdlg); end;



pause  % Press any key.

clc

% The 'magnifying glass' icon, when clicked, enters "zoom" mode, such that
% clicks in the plot window half the axes limits, centered around that point.
% (An alt-click (option-click and control-click on the Mac) zooms out.)

% These functions can also be selected from the MRMVTool menu.

% For example, to zoom in twice:

mrmvtool('zoom','in'); drawnow; mrmvtool('zoom','in');


pause  % Press any key.


% The axes limits can be restored to their original values also

mrmvtool('zoom','restore');



pause  % Press any key.

clc

% Now compute the reciprocal modal vector matrix (RMVs)
% (click the 'Compute RMVs' button)

eval(mrmvtool('compute'));


% And display the filtered response
% (select 'Filtered Data' from the popup menu and click the 'Plot' button)

mrmvtool('popup','Filtered Data');
eval(mrmvtool('plot'));



pause  % Press any key.
```

mrmvtool_demo.m — Demo Script for MRMVTool (cont.)

```
clc

% The filtered data can be saved
% (type the desired variable name in the box next to the 'to workspace as'
% button, then click the button)

mrmvtool('setstr','toworkspace','ufrfs');
eval(mrmvtool('toworkspace'));



pause  % Press any key.

clc

% The RMV matrix can be plotted
% (select 'RMV Matrix' from the popup menu and click the 'Plot' button)

mrmvtool('popup','RMV Matrix');
eval(mrmvtool('plot'));



pause  % Press any key.
if strcmp(get(gcf,'Name'),'MRMVTool Warning'),
    eval(get(findobj(gcf,'String','OK'),'CallBack'));
end;

clc

% And the RMV matrix can also be saved out to the workspace
% (type the desired variable name in the box next to the 'to workspace as'
% button, then click the button)

mrmvtool('setstr','toworkspace','rmv_matrix');
eval(mrmvtool('toworkspace'));


% Here is its value

rmv_matrix



pause  % Press any key.

clc

% The RMVs can be restricted to have only real numbers by setting the
% appropriate checkbox in the Options dialog.

% First, open the Options dialog
% (click the 'Options...' button)
mrmvtool('options');

pause  % Press any key

% Now check the 'Restrict RMVs to Real Numbers' box
mrmvtool('options','realrmv',1);

pause  % Press any key

% And close the dialog box
% (click the 'OK' button)
mrmvtool('options','OK');
```

mrmvtool_demo.m — Demo Script for MRMVTool (cont.)

```
pause  % Press any key.

clc

% The RMVs can then be recomputed
% (click the 'Compute RMVs' button)

eval(mrmvtool('compute'));


% And its value extracted

mrmvtool('popup','RMV Matrix');
mrmvtool('setstr','toworkspace','rmv_matrix');
eval(mrmvtool('toworkspace'));
rmv_matrix



pause  % Press any key.

clc

% The spectral lines used to compute the RMVs can be restricted by
% changing the 'Freq. Indices' value.  Let us set it to those frequencies
% within 10% +/- of the actual natural frequencies.

mask = zeros(size(frqs));
for k=1:length(W),   mask = mask | (frqs>=0.9*W(k) & frqs<=1.1*W(k));   end;
indices = find(mask);

mrmvtool('setstr','indices','indices');
mrmvtool('popup','Unfiltered Data');
eval(mrmvtool('plot'));

% And recompute the RMVs and extract the RMV matrix

eval(mrmvtool('compute'));
mrmvtool('popup','RMV Matrix');
mrmvtool('setstr','toworkspace','rmv_matrix');
eval(mrmvtool('toworkspace'));
rmv_matrix



pause  % Press any key.

clc

% The spectral lines used to compute the RMVs can be chosen by clicking in the
% plot window if the Unfiltered or Filtered Data has been plotted.

mrmvtool('popup','Unfiltered Data'); eval(mrmvtool('plot'));
ax1=findobj(mrmvtool('getfig'),'Tag','ax1'); xlims=get(ax1,'XLim'); ylims=get(ax1,'YLim');
if strcmp(get(ax1,'YScale'),'log'), ycenter=sqrt(prod(ylims)); else, ycenter=sum(ylims)/2; end;
mrmvtool('setstr','indices',num2str(round(length(frqs)/2)));

mrmvtool('choose',eval(mrmvtool('getstr','indices'),'''err'''),[sum(xlims)/2 ycenter]);

pause  % Press any key.

% Let's choose 10 points (at random) to add/remove

xislog=strcmp(get(ax1,'XScale'),'log'); if (xislog),xlims=log(xlims);end;
for k=1:10,
```

mrmvtool_demo.m — Demo Script for MRMVTool (cont.)

```
    pt=sum(([0 1]+[1 -1]*rand).*xlims);
    if (xislog), pt=exp(pt); end;
    mrmvtool('choose',eval(mrmvtool('getstr','indices'),'''err'''),[pt ycenter]);
    drawnow;
end;



pause  % Press any key.

clc

% The RMVs can be used to filter a new response


% The same system will be used, but with some multiplicative noise.

frfs2 = frfs.*(1+abs(randn(size(frfs)))/10.*exp(sqrt(-1)*rand(size(frfs))*2*pi));


% This unfiltered, noisy, response can be plotted
% (type 'frfs2' in the 'Unfiltered Data' text box, select 'Unfiltered Data'
% in the popup menu and click the 'Plot' button)

mrmvtool('setstr','unfiltered','frfs2');
mrmvtool('popup','Unfiltered Data');
eval(mrmvtool('plot'));



pause  % Press any key.

clc

% The same RMVs can then be used to produce the filtered FRFs
% from the new (noisy) transfer functions
% (select 'Filtered Data' in the popup menu and click the 'Plot' button)

mrmvtool('popup','Filtered Data');
eval(mrmvtool('plot'));



pause  % Press any key.

clc

% The GUI is closed when we are finished
% (click the 'Exit' button)

mrmvtool('exit');



% This demo is over.
echo off
```

8.1.5 `ndof.m` — Simple n Degree of Freedom Systems

`ndof` produces the state-space representation of an n degree of freedom system that is used in evaluating the MRMV method and various other system identification algorithms. It is designed to be able to return continuous- or discrete-time models, in addition to giving the exact reciprocal modal vectors computed from the configuration-space mass and stiffness matrices. Further options allow simulation of velocity and acceleration output and varying structural characteristics.

```
function [a,b,c,d,M,C,K,PP,l,W,Z,U,V] = ndof(n,t,outtype,massfract,cfract,kfract)
% NDOF set up state-space representation of an N degree-of-freedom system.
%
%   [A,B,C,D] = NDOF(N) returns the state-space matrices for an N degree-
%                       of-freedom system.  It is a chain of N identical
%                       masses, connected to each other with identical
%                       spring/damper couplings like a train, with the
%                       first mass also connected with the same coupling
%                       to a wall.  The outputs of the system are the
%                       displacements of each mass.
%
%              |
%              |    k    +---+    k    +---+        +---+    k    +---+
%              |--/\/\/--|   |--/\/\/--|   |        |   |--/\/\/--|   |
%              |----]----| m |----]----| m |  ...   | m |----]----| m |
%              |    c    +---+    c    +---+        +---+    c    +---+
%              |           o o          o o          o o          o o
%              +-----------------------------------------------------
%
%   [A,B,C,D,MM,CC,KK,PP,L,W,Z,U,V] = NDOF(N) also returns:
%
%           MM,CV,KK  mass, damping, and stiffness matrices
%           PP        inverse of the configuration-space eigenmatrix;
%                     it is the exact reciprocal modal filter matrix,
%                     such that PP*measured = modal
%                     (sorted by decreasing freq.)
%           U,V,L     the state-space eigenmatrix, its transposed inverse,
%                     and eigenvalues (all sorted by decreasing frequency)
%           W,Z       modal natural frequency (rad/sec) and damping ratios
%                     (assuming the damping decouples)
%
%   NDOF(N,T) returns (A,B,C,D) in discrete-time with sample time T.
%             T=0 implies continuous-time.
%
%   NDOF(N,T,OUTTYPE) uses OUTTYPE as the outputs of the system.  The default
%                     is 'displacement'; other valid choices are 'velocity',
%                     and 'acceleration'.
%
%   NDOF(N,T,OUTTYPE,MASSFRACT) makes the mass MASSFRACT (all if MASSFRACT
%                    is a scalar; MASSFRACT(i) for the i-th mass if the
%                    length is N).
%
%   NDOF(N,T,OUTTYPE,MASSFRACT,DAMPFRACT,STIFFFRACT) does the same with
%                    damping and stiffness.

% Copyright (c)1996, Erik A. Johnson <johnsone@uiuc.edu>, 4/10/96

% check args
if (nargin<1),
    error('NDOF requires at least one argument, the # of degrees-of-freedom.');
elseif (nargin>6),
    error('NDOF takes at most 6 arguments.');
end;
if (nargin<2),
    t=0;
elseif (length(t)>1),
    error('NDOF requires that the sampling time T be a scalar');
elseif (~all(size(t))),
    t=0;
elseif (t<0), %t<0 could be a flag for something later
    t=abs(t);
```

ndof.m — Simple n Degree of Freedom Systems (cont.)

```
end;
if (nargin<3), outtype=[]; end;
if (~all(size(outtype))), outtype='displacement'; end;
if (nargin<4), massfract=[]; end;
if (nargin<5), cfract   =[]; end;
if (nargin<6), kfract   =[]; end;

% do the fractions
if (~all(size(massfract))), massfract=ones(n,1);
elseif (length(massfract)==1), massfract=massfract*ones(n,1);
elseif (length(massfract)==n), massfract=massfract(:);
else, error('NDOF requires that MASSFRACT be scalar or N-by-1.');
end;
if (~all(size(cfract))), cfract=ones(n,1);
elseif (length(cfract)==1), cfract=cfract*ones(n,1);
elseif (length(cfract)==n), cfract=cfract(:);
else, error('NDOF requires that CFRACT be scalar or N-by-1.');
end;
if (~all(size(kfract))), kfract=ones(n,1);
elseif (length(kfract)==1), kfract=kfract*ones(n,1);
elseif (length(kfract)==n), kfract=kfract(:);
else, error('NDOF requires that KFRACT be scalar or N-by-1.');
end;

m1= 1 * massfract;
c1=.1 * cfract;
k1= 1 * kfract;

M=diag(m1);
C=diag(c1) + diag([c1(2:n);0]);
K=diag(k1) + diag([k1(2:n);0]);
if (n>1),
    C=C-diag(c1(2:n),1)-diag(c1(2:n),-1);
    K=K-diag(k1(2:n),1)-diag(k1(2:n),-1);
end;

% state-space
a=[zeros(n) eye(n); -inv(M)*K -inv(M)*C];
b=[zeros(n);inv(M)];
if (lower(outtype(1))=='v'),
    c=[zeros(n) eye(n)];
    d=zeros(n);
elseif (lower(outtype(1))=='a'),
    c=-[inv(M)*K inv(M)*C];
    d=eye(n);
elseif (lower(outtype(1))=='d'),
    c=[eye(n) zeros(n)];
    d=zeros(n);
else,
    error(['An OUTTYPE value of `' outtype(:).' `' is not valid.']);
end;

% discrete-time?
if (t~=0), [a,b]=c2d(a,b,t); end;

if (nargout>7),
    % principal coordinates (Craig, pp. 341ff), sorted by decreasing frequency
    [PP,WW]=eig(inv(M)*K);
    [WW,k]=sort(diag(WW));
    PP = PP(:,k(n:-1:1));
    W = sqrt(WW(n:-1:1));
    CC = PP' * C * PP;
    Z = diag(CC)/2./W;
    PP=inv(PP); %note that PP is now the exact modal filter matrix,
                %such that PP*(n-by-1 measured resp) = modal resp
    % check off-diagonals in damping to see if we decoupled
    CC = (CC - diag(diag(CC)))/norm(CC);
    if (any(abs(CC(:))>eps*100)),
        disp('NDOF:  WARNING:  damping did not decouple!');
    end;
```

ndof.m — Simple n Degree of Freedom Systems (cont.)

```
    % find eigenvalues; rearrange, grouping first of conjugate pairs at beginning
    [U,L]=eig(a);
    U=[U(:,1:2:length(U)) U(:,2:2:length(U))];
    l=diag(L); L=diag([l(1:2:length(l));l(2:2:length(l))]); l=diag(L);
    V=inv(U).';
end;

%% example of how to do a time response using the output of ndof
%n=3;
%[a,b,c,d] = ndof(n);
%t=(0:.05:250)';
%u=randn(length(t),n);
%[y,x]=lsim(a,b,c,d,u,t);
%
%u2=randn(length(t),n);
%[y2,x2]=lsim(a,b,c,d,u2,t);
```

8.1.6 normv.m — Compute Norm of Column Vectors

normv computes the norm of each column of a matrix. It functions identically to the standard norm function for a vector argument, but for matrices, norm produces a norm of the entire matrix, whereas this function computes the norm of each column, returning a row vector with as many elements as the matrix had columns.

```
function out = normv(X,P)
% NORMV  Norm of vector or of each column of matrix.
%
%  NORMV(X,P) acts identically to NORM(X,P) if X is a vector (row or column),
%             but if X is a matrix, NORMV performs NORM on each column of X.
%             In other words, Y=NORMV(X,P) is the same as
%
%                for k=1:size(X,2), Y(1,k)=NORM(X(:,k),P); end;
%
%  See also NORM.

% Copyright (c)1995, Erik A. Johnson <johnsone@uxh.cso.uiuc.edu>, 7/26/95


if (size(X,1)<=1)|(size(X,2)<=1),
    if nargin<2,
        out = norm(X);
    else,
        out = norm(X,P);
    end;

else,

    if nargin<2,
        P = 2;
    elseif isstr(P),
        if strcmp(P,'inf'),
            P = inf;
        elseif strcmp(P,'fro'),
            P = 2;
        elseif strcmp(P,'-inf'),
            P = -inf;
        else,
            error('Invalid P string in NORM2(X,P).');
        end;
    elseif (size(P,1)~=1)|(size(P,2)~=1),
        error('P in NORM2(X,P) must be a scalar.');
    end;

    if (P==2),
        out = sqrt(sum(abs(X).^2));
    elseif (P==1),
        out = sum(abs(X));
    elseif (P==inf),
        out = max(abs(X));
    elseif (P==-inf),
        out = min(abs(X));
    else,
        out = sum(abs(X).^P).^(1/P);
    end;

end;
```

8.1.7 `sbys2stack.m` — Stack Side-by-Side Blocks

`sbys2stack` takes a matrix with blocks that are side-by-side and returns a matrix of those blocks stacked on top of each other. It is the converse of `stack2sbys`.

```
function y = sbys2stack(x,N)
% SBYS2STACK  Stack blocks in a matrix.
%
%   SBYS2STACK(X,N) takes the N side-by-side blocks in X and stacks them.
%                   In other words, if X is m-by-(N*n), and X1,X2,...,XN
%                   are m-by-n, and X = [X1 X2 X3 ... XN], then the output
%                   is [X1;X2;X3;...;XN].
%
%   See also STACK2SBYS, RESHAPE.

% Copyright (c) 1995, Erik A. Johnson <johnsone@uxh.cso.uiuc.edu>, 3/22/95

error(nargchk(2,2,nargin));

if (isempty(x)), y=x.'; return; end; %added this line, 3/22/95

[m,n]=size(x);
if (rem(n,N)~=0), error('The number of columns in X must be a multiple of N.'); end;
y=zeros(m*N,n/N);
[ii,jj]=meshgrid(((1:N)-1)*n*m/N,1:m);
[ii,jj]=meshgrid(((1:(n/N))-1)*m,ii(:)+jj(:));
y(:)=x(ii(:)+jj(:));
```

8.1.8 `stack2sbys.m` — Unstack Blocks to Side-by-Side

`stack2sbys` takes a matrix with blocks that are stacked on each other and returns a matrix of those blocks side-by-side. It is the converse of `stack2sbys`.

```
function y = stack2sbys(x,N)
% STACK2SBYS  Place blocks side-by-side in a matrix.
%
%   STACK2SBYS(X,N) takes the N stacked blocks in X and places them side-by-side.
%                   In other words, if X is (N*m)-by-n, and X1,X2,...,XN
%                   are m-by-n, and X = [X1;X2;X3;...;XN], then the output
%                   is [X1 X2 X3 ... XN].
%
%   See also SBYS2STACK, RESHAPE.

% Copyright (c) 1995, Erik A. Johnson <johnsone@uxh.cso.uiuc.edu>, 3/22/95

error(nargchk(2,2,nargin));

if (isempty(x)), y=x.'; return; end; %added this line, 3/22/95

[m,n]=size(x);
if (rem(m,N)~=0), error('The number of rows in X must be a multiple of N.'); end;
y=zeros(m/N,n*N);
[ii,jj]=meshgrid(((1:n)-1)*m,1:(m/N));
[ii,jj]=meshgrid(((1:N)-1)*(m/N),ii(:)+jj(:));
y(:)=x(ii(:)+jj(:));
```

8.1.9 `str2strmat.m` — String Conversion Utility

`str2strmat` is a utility function that converts a string vector that contains newline or return characters into a string matrix with one row per newline- or return-separated segment. It is used to construct the legend text in `mrmv_test_adapt`.

```
function [out,ll] = str2strmat(s,st,en)
% STR2STRMAT  Convert a string to a matrix of strings.
%
%   STR2STRMAT(STRING) takes as input a STRING of characters with embedded
%                      newline (or return) characters, and returns a matrix
%                      with each row being a line from the original string.
%                      The input STRING is assumed to be a vector, not a
%                      matrix.  (Zero-padding is used for lines shorter
%                      than the longest.  The newline/return character is
%                      NOT included in the output.)
%
%   STR2STRMAT(STRING,C) does the same, but uses the character C as the
%                        line separator.  (The character C is NOT included
%                        in the output.)
%
%   STR2STRMAT(STRING,STARTINDEX,ENDINDEX) does the same, but rather than
%                        using a particular character as a marker for line
%                        endings, this form specifically gives a pair of
%                        vectors, STARTINDEX and ENDINDEX (that should be
%                        the same size and shape), that give the starting
%                        and ending indices, respectively, into the STRING
%                        for each line.
%
%   [STRMAT,LINELEN] = STR2STRMAT(...) returns both the string matrix and
%                        a column vector of line lengths.

% Copyright (c)1995, Erik A. Johnson <johnsone@uxh.cso.uiuc.edu>, 8/29/95

% check # of args
if (nargin<1), error('STR2STRMAT requires at least one input argument.');
elseif (nargout>1), error('STR2STRMAT produces only one output argument.');
end;

% if empty, return empty
if (~all(size(s))), out=[]; ll=[]; return; end;

% compute st and en if necessary
s = s(:)';
if (nargin<3),
    if (nargin<2), c=sprintf('\n'); else, c=st; end;
    en=find(s==c)-1;
    if (length(en)==0),
        en=length(s);
    elseif (en(length(en))+1<length(s)),
        en(length(en)+1)=length(s);
    end;
    st = [1 en(1:length(en)-1)+2];
else,
    en = en(:)';
    st = st(:)';
    if (length(en)~=length(st)),
        error('STR2STRMAT requires that the start and end index matrices be the same size.');
    end;
end;

% allocate a matrix
ll=en-st+1;
nlines = length(ll);
out=setstr(zeros(max(ll),nlines));

% calculate index into new matrix
newi = ones(1,sum(ll));
newi(cumsum(ll)) = 1+max(ll)-ll;
newi = cumsum([1 newi(1:length(newi)-1)])';
```

str2strmat.m — String Conversion Utility (cont.)

```
% calculate index into old matrix
oldi = ones(1,sum(ll));
oldi(cumsum(ll(1:nlines-1))+1) = st(2:nlines)-en(1:nlines-1);
oldi(1) = st(1);
oldi = cumsum(oldi);

% do the transfer
out(newi) = s(oldi);
out = out';
```

8.2 H_∞-BASED IDENTIFICATION CODES

8.2.1 `hinfid.m` — H_∞-based Identification

`hinfid` does H_∞-based system identification of a single-input, multi-output (SIMO) system using pulse response data. Linear and nonlinear algorithms are available, as well as various window functions. The resulting output is numerator and denominator polynomial coefficients.

```
function [num,den,bound] = hinfid(pulseresp,no,type,n,wind,m,roots_tolerance)
% HINFID  Identify SIMO system via H-infinity identification.
%
%   [NUM,DEN,BOUND] = HINFID(H,NO,TYPE,n,WIND,m) does an H-infinity
%                                                identification, where
%         H     is the pulse responses (one output per column) of a
%               number of single-input, multi-output (SIMO) systems.
%         NO    is the number of outputs (default is 1).
%         TYPE  is either 'linear' or 'nonlinear' (default), specifying
%               whether to use the simple linear algorithm or the two-
%               stage nonlinear algorithm.
%         n     is the half-window size (default is half #rows(H))
%         WIND  is the type of windowing function to use; valid choices are:
%                  'boxcar' (the default)    (1 in |k|<=n)
%                  'triang' or 'triangular'  (1-|k|/n)
%                  'trap'   or 'trapezoidal' (1 in 0<=k<=2M, >0 in |k-m|<n)
%                  'cos'    or 'cosine'      (cos(k*pi/(2N+1))
%                  'hamming'                 (.54+.46*cos(k*pi/n))
%                  'spline'                  ([sin(k*pi/m)*m/(k*pi)]^2)
%                  'hanning'                 ([1+cos(k*pi/(n+1))]/2)
%                  'blackman'                (.42+.5*cos(k*pi/n)+.08*cos(2*k*pi/n))
%                  'bartlett'                same as 'triangular'
%                  'none'                    uses 'boxcar'
%         m     is an optional auxiliary variable used by the 'trapezoidal'
%               (default value is n/2) and 'spline' (default value size(H,1))
%         NUM   is the numerator polynomial of the identified model,
%               one row per column of the pulse response H.
%         DEN   is the denominator polynomial of the identified model,
%               one row per column of the pulse response H.
%         BOUND is the estimate of the H-infinity identification error bound
%               (for the nonlinear algorithm only).  Its size is the number
%               of columns of H divided by NO.
%
%   Note:  n and m must be scalar or vectors, but length(n), length(m),
%          and size(H,2)/NO must all be the same or any of them can be 1.
%          In other words, if L=[length(n) length(m) size(H,2)/NO] then
%          all(L==max(L)|L<=1) must be true.
%
%   BOUND = HINFID(H,NO,TYPE,n,WIND,m) simply returns the identification
%                                      error bound.
%
%   ... = HINFID(H,NO,TYPE,n,WIND,m,TOL) sets the tolerance passed to
%                                        minreal to factor out common
%         numerator and denominator roots.  Pass a negative value to
%         only remove common roots at the origin.  The default value
%         used by minreal is 10*abs(root)*sqrt(eps).
%
%         This is generally required for large n since root solving
%         for large polynomials (e.g., order greater than a couple
%         hundred) is quite time consuming and may introduce additional
%         error.
%
%   [K,W] = HINFID(H,NO,'wind',n,WIND,m) returns the window function W(K,n).

% Copyright (c)1996, Erik A. Johnson <johnsone@uiuc.edu>, 6/16/96

% check # of args
if (nargin<5), error('HINFID requires at least 5 input arguments.');
elseif (nargin>7), error('HINFID takes at most 7 input arguments.');
elseif (nargout>3), error('HINFID produces at most 3 output arguments.');
end;
```

hinfid.m — H∞-based Identification (cont.)

```
% handle default values
if (isempty(no)), no=1; end;
if (isempty(type)), type='nonlinear'; end;
if (isstr(type)),
    type_orig = type(:)';
    type = lower([type_orig ' ']);
    if (~any(type(1)=='lnw')),
        error(['HINFID does not recognize ''' type_orig ''' as a valid ID TYPE.']);
    end;
else,
    error('HINFID requires that the ID TYPE be a string.');
end;
if (isempty(pulseresp)),
    if (type(1)~='w'),
        error('HINFID needs a non-empty pulse response matrix.');
    else,
        pulseresp = ones(4*max(n(:))+1,1);
    end;
end;
if (isempty(n)), n=floor(N/2); end;
if (isempty(wind)), wind='boxcar'; end;
if (nargin<6), m=[]; end;
if (nargin<7), roots_tolerance=[]; end;

% get some size information and coerce everything to the right size
[N,M] = size(pulseresp);
if (rem(M,no)~=0),
    error('HINFID requires that the # of columns in H be a multiple of NO.');
end;
M = M / no;
if any(2*n(:)>=N), error('HINFID requires that n be less than half size(H,1).'); end;
if (type(1)=='1'),
    k = (0:N-1)';
else,
    k = [0:N/2 -floor((N-1)/2):-1]';
end;
m=m(:)'; if (isempty(m)), m=[]; end;
n=n(:)';
L = [length(m) length(n) M];
if (any(L~=max(L) & L>1)),
    error(['HINFID requires that length(n), length(m), and size(H,2)/NO ' ...
           'be the same or 1.']);
end;
ii_pr = 1:size(pulseresp,2);
if (M<max(L)),
    M = max(L);
    ii_pr = ii_pr.';
    ii_pr = ii_pr(:,ones(1,M));
    ii_pr = ii_pr(:)';
end;
if (length(m)>0 & length(m)<M),
    m = m*ones(1,M);
end;
if (length(n)>0 & length(n)<M),
    n = n*ones(1,M);
end;
if (M>1), k=k(:,ones(1,M)); end;
if (N>1),
    if (~isempty(m)), m=m(ones(N,1),:); end;
    n = n(ones(N,1),:);
end;
% expand for NO>1
if (no>1),
    ii = 1:M;
    ii = ii(ones(no,1),:);
    ii = ii(:)';
    if (~isempty(m)), m=m(:,ii); end;
    n = n(:,ii);
    k = k(:,ii);
    M = M * no;
```

hinfid.m — H∞-based Identification (cont.)

```
end;

% handle the different window functions
wind_orig = wind(:)';
wind = lower([wind_orig '    ']);
wind = wind(1:3);
if (all(wind=='non')),
    wind = 'box';
elseif (all(wind=='bar')),
    wind = 'tri';
end;
if (all(wind=='box')),
    weight = (abs(k) <= n);
elseif (all(wind=='tri')),
    weight = max(1-abs(k)./n,0);
elseif (all(wind=='tra')),
    if (isempty(m)), m=floor(n/2); end;
    if (any(m(:)<0) | any(m(:)>n(:))),
        error('HINFID with trapezoidal windows requires 0 <= m <= n.');
    end;
    weight = min(1,max((n-abs(k-m))./(n-m+(n==m)),0));
    if (any(n==m&k==n)),
        ii = find(n==m&k==n);
        weight(ii) = ones(length(ii),1);
    end;
    n = n + m; %because the trapezoidal window is skewed toward causal values
    if any(2*n(:)>=N),
        error(['HINFID with a trapezoidal window requires that n+m be ' ...
                'less than half size(H,1).']);
    end;
elseif (all(wind=='cos')),
    weight = cos(pi*k./(2*n+1)) .* (abs(k)<=n);
elseif (all(wind=='ham')),
    weight = (.54+.46*cos(pi*k./n)) .* (abs(k)<=n);
elseif (all(wind=='spl')),
    if (isempty(m)), m=N; end;
    weight = (sin(k*pi./m).*m./(k+(k==0))/pi).^2 .* (abs(k)<=n);
    if (any(k(:)==0)),
        weight(k==0) = weight(k==0) + (abs(k(k==0))<=n(k==0));
    end;
elseif (all(wind=='han')),
    weight = (1+cos(pi*k./(n+1)))/2 .* (abs(k)<=n);
elseif (all(wind=='bla')),
    weight = (.42+.5*cos(pi*k./n)+.08*cos(2*pi*k./n)) .* (abs(k)<=n);
else,
    error(['HINFID does not recognize ''' wind_orig ...
           ''' as a valid windowing function.']);
end;

% handle window function version
if (type(1)=='w'),
    if (N>1),
        [k,ii] = sort(k);
        weight = weight(ii,:);
    end;
    num = k;
    den = weight;
    return;
end;

% reduce n back to a row vector
n = n(1,:);
clear('k','m');

% weight the pulse response
pulseresp = pulseresp(:,ii_pr) .* weight;

if (type(1)=='l'), % handle linear identification

    % get rid of ending 0 terms
```

hinfid.m — H∞-based Identification (cont.)

```
    mask = (n>0 & pulseresp(n+1+(0:length(n)-1)*M)==0);
    while (any(mask)),
        n(mask) = n(mask) - 1;
        mask = (n>0 & pulseresp(n+1+(0:length(n)-1)*N)==0);
    end;

    % make some space
    max_n_plus_1 = max(n)+1;
    num = zeros(M,max_n_plus_1);
    den = num;

    % compute it
    den((1:length(n))+M*(max_n_plus_1-(n+1))) = ones(M,1);
    for i=1:M,
        num(i,max_n_plus_1-n(i):max_n_plus_1) = pulseresp(1:n(i)+1,i)';
    end;

    % shorten if possible
    while (all(num(:,1)==0 & den(:,1)==0)),
        num(:,1)=[];
        den(:,1)=[];
    end;

    % no bound for linear algorithm
    bound = [];

else, % handle nonlinear identification

    % make some space
    num = zeros(M,4*max(n)+1);
    den = num;
    bound = zeros(1,M/no);

    % compute it
    for i=1:no:M,
        % extract the data for this response
        hw_anticausal = pulseresp(N:-1:N-n(i)+1,i:i+no-1)';
        hw_causal     = pulseresp(1:n(i)+1,i:i+no-1)';

        % eigenproblem of sum of square of Hankel matrices
        hank2 = zeros(n(i));
        for j=1:no,
            hank2 = hank2 + hankel(hw_anticausal(j,:))^2;
        end;
        [X,Sigma2] = eig(hank2);
        [junk,ii] = max(diag(Sigma2));
        x1     = X(:,ii);
        sigma1 = sqrt(Sigma2(ii,ii));

        % compute the polynomials
        for j=1:no,
            Lambda_squiggle = hankel([hw_anticausal(j,n(i)-1:-1:1) hw_causal(j,:)]);
            Lambda_squiggle = Lambda_squiggle(n(i):-1:1,:);
            % compute the estimated model
            num1 = x1'*Lambda_squiggle;
            den1 = [x1' zeros(1,n(i))];

            % remove common roots
            if (isempty(roots_tolerance)),
                [num1,den1] = minreal(num1,den1);
            elseif (roots_tolerance(1)>=0),
                [num1,den1] = minreal(num1,den1,roots_tolerance(1));
            else,
                num_zero_roots = min(sum(cumprod(fliplr(num1==0))), ...
                                     sum(cumprod(fliplr(den1==0))));
                if (num_zero_roots>0),
                    num1(length(num1)+(1-num_zero_roots:0)) = [];
                    den1(length(den1)+(1-num_zero_roots:0)) = [];
                end;
            end;
```

hinfid.m — H∞-based Identification (cont.)

```
            % insert into outputs
            num(i-1+j,size(num,2)+(1-length(num1):0)) = num1;
            den(i-1+j,size(den,2)+(1-length(den1):0)) = den1;
        end;

        bound((i-1+no)/no) = sigma1;
    end;

    % shorten outputs if possible
    while (all(num(:,1)==0 & den(:,1)==0)),
        num(:,1)=[];
        den(:,1)=[];
        if (isempty(num) | isempty(den)), break; end;
    end;

end;

if (nargout<=1),
    num = bound;
end;
```

8.2.2 `hinfid_test.m` — Quick Test of `hinfid` H_∞-based Identification

`hinfid_test` does a quick test of the H_∞ identification function `hinfid`. It attempts to identify the system $H(z) = (3z^2 + 3) / (5z^2 + 2z + 1)$ in the presence of noise using linear and nonlinear algorithms with various window functions.

```
% hinfid_test.m

% a very quick test of the H-infinity identification

num=[3 0 3]; den=[5 2 1];     % base system: H(z)=(3z^2+3)/(5z^2+2z+1)
N=512;                        % number of data points to use
n=20;                         % order of the identification
epsilon=0.1;                  % magnitude of the noise

% identify via linear and nonlinear algorithms with various windows
types=str2mat('linear','nonlinear');
windows=str2mat('boxcar','triangular','trapezoidal','cosine', ...
                'hamming','spline','hanning','blackman');

% simulate the original system
[H_exact,omega]=freqz(num,den,N,'whole');
h_exact = real(ifft(H_exact));

% compute some noise
rand('seed',21217); % so we can repeat this exactly
Noise(1:N/2+1,1) = [0;exp(sqrt(-1)*rand(N/2-1,1)*2*pi);sign(rand-.5)];
Noise(N/2+2:N,1) = conj(Noise(N/2:-1:2,1));

% add the noise to the base system
H_noisy = H_exact + epsilon*Noise;
h_noisy = real(ifft(H_noisy));

% create some space
nums = zeros(length(n)*size(types,1)*size(windows,1),3*n);
dens = nums;
H = zeros(N,size(nums,1));
bounds = zeros(1,size(nums,1));

% initialize some graphics
clf('reset');
h = plot([0;0],H(1:2,:),'k-');
set(gca,'XLim',[0 1]);
xlabel('frequency_in_radians*T/pi'); ylabel('magnitude of model error');
title('H-infinity ID of H(z)=(3z^2+3)/(5z^2+2z+1), noise magnitude 0.1');
co = get(gca,'ColorOrder');
co(all(co'==1),:) = [];
co=[1 1 1;co];
if (size(co,1)>1 & size(co,1)<size(windows,1)),
    extras = size(windows,1) - size(co,1);
    co = [co; (1:extras)'/(extras+1)*[1 1 1]];
end;
styles = str2mat('-','--',':','-.');
hwaitbar = waitbar(0,'Doing identification ...');

% loop through all configurations
for i=1:size(types,1),
    this_style = deblank(styles(rem(i-1,size(styles,1))+1,:));
    for j=1:size(windows,1),
        % do the identification
        [num1,den1,b] = hinfid(h_noisy,1,deblank(types(i,:)),n,deblank(windows(j,:)));
        % insert into big storage
        irows = (1:length(n))+length(n)*((i-1)*size(windows,1)+j-1);
        nums(irows,size(nums,2)+(1-size(num1,2):0)) = num1;
        dens(irows,size(dens,2)+(1-size(den1,2):0)) = den1;
        if (~isempty(b)), bounds(1,irows)=b; end;
        % compute and plot the magnitude of the model error in freq. domain
        for k=1:length(n),
            H(:,irows(k)) = freqz(num1,den1,N,'whole');
            set(h(irows(k)),'XData',omega/pi,'YData',abs(H(:,irows(k))-H_exact)');
        end;
```

hinfid_test.m — Quick Test of hinfid H∞-based Identification (cont.)

```
        this_color = co(rem(j-1,size(co,1))+1,:);
        set(h(irows),'LineStyle',this_style,'Color',this_color);
        waitbar((i-1+j/size(windows,1))/size(types,1));
    end;
end;
close(hwaitbar);
legend(h([1+(0:size(types,1)-1)*size(windows,1)*length(n) ...
          1+(0:size(windows,1)-1)*length(n)]), ...
       str2mat(upper(types),windows));
```

8.2.3 `hinfid_example1.m` — Example I of H_∞-based Identification

`hinfid_example1` runs the first example of H_∞-based identification. Its task is the identification of the simple single-input, single-output system $H(z) = (3z^2 + 3) / (5z^2 + 2z + 1)$.

```
% hinfid_example1.m
echo on

% This runs the first H-infinity identification example
%
% The system is the discrete-time system H(z)=(3z^2+1)/(5z^2+2z+1)



% load in past data if already run
datafile = 'hinfid_ex1';
eval('load(datafile);','comp=1;');


% set up some variables
scrn = 1;  % change to 0 to do hardcopies

if (scrn),
    g1 = 1;
    g2 = -[1 1 1];
else,
    g1 = 0;
    g2 = [1 1 1];
end;


% set line colors, styles, and widths
linestyle=str2mat('x',':','-.','--','-'); linestyle=str2mat('-',linestyle,linestyle);
linewidth=  [16;   4 ; 7 ; 1 ; 1 ; 1 ;     6 ; 2 ; 8 ; 8 ; 8 ]/8;
markersize= [ 6;  2.5; 6 ; 6 ; 6 ; 6 ;    2.5; 6 ; 6 ; 6 ; 6 ];
graylevel=1-[ 1;   1 ; 1 ; 1 ; 1 ; 1 ;     .4; 1 ; .4; .4; .4];
mask=(graylevel==0|graylevel==1|scrn);
graylevel(mask) = (1-graylevel(mask));
graylevel = graylevel(:,[1 1 1]);




%%%%%%%%%%%%%%%%%%%%%%%%%
%                       %
%  windowing functions  %
%                       %
%%%%%%%%%%%%%%%%%%%%%%%%%
nn=20; m=floor(nn/4); NN=4*nn;
[Kbox,Wbox] = hinfid(ones(NN,1),1,'wind',nn,'boxcar');
[Ktri,Wtri] = hinfid(ones(NN,1),1,'wind',nn,'triang');
[Ktra,Wtra] = hinfid(ones(NN,1),1,'wind',nn,'trapez',m);
[Kham,Wham] = hinfid(ones(NN,1),1,'wind',nn,'hammin');
[Kspl,Wspl] = hinfid(ones(NN,1),1,'wind',nn,'spline',NN);
[Kcos,Wcos] = hinfid(ones(NN,1),1,'wind',nn,'cosine');
clf('reset');
h=plot(Kbox/nn,Wbox,'w-',   Ktri/nn,Wtri,'w--',  Ktra/nn,Wtra,'wo', ...
       Kham/nn,Wham,'w:',   Kspl/nn,Wspl,'w-.',  Kcos/nn,Wcos,'wx');
axis([-1.5 1.5 -.1 1.1]);
set(h([3 6]),'MarkerSize',4);
sxlabel('\times {\i k}/{\i n}'); sylabel('\times weight');
labs = str2mat('boxcar','triangular',['trapezoidal ({\i m}=' num2str(m) ')'], ...
               'Hamming',['spline ({\i M}=' num2str(NN) ')'],'cosine');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
[hax,hli,hte] = slegend([.54 .3],h,labs);
fn = 'hinfid_ex1_0_windows';
if (~scrn), drawnow; printsto('-deps',[fn '.eps']); end;
stitle(['\times {\i H}_{\infty} Identification Windowing Functions ({\i n}=' ...
        num2str(nn) ', {\i N}=' num2str(NN) ')']);
drawnow; if (scrn), pause; else, printsto('-dps',[fn '.ps']); end;
```

hinfid_example1.m — Example I of H∞-based Identification (cont.)

```
%%%%%%%%%%%%%%%
%             %
%  Example I  %
%             %
%%%%%%%%%%%%%%%
if (comp),
    N = 512;                    % number of points in response; must be even
    n = [5 10 20 40 80];        % model orders
    epsilon = 0.1;              % noise magnitude in frequency domain

    % simulate the original system
    num=[3 0 3]; den=[5 2 1];
    [H_exact,omega]=freqz(num,den,N,'whole');
    rand('seed',21217); % so we can repeat this exactly
    Noise(1:N/2+1,1) = [0;exp(sqrt(-1)*rand(N/2-1,1)*2*pi);sign(rand-.5)];
    Noise(N/2+2:N,1) = conj(Noise(N/2:-1:2,1));
    H_noisy = H_exact + epsilon*Noise;
    h_exact = real(ifft(H_exact));
    h_noisy = real(ifft(H_noisy));

    % compute pulse responses
    [t,h_exact_stairs] = stairs(h_exact);
    [t,h_noisy_stairs] = stairs(h_noisy);

    % do the identification
    [num_lin,den_lin]             = hinfid(h_noisy,1,'linear'   ,n,'triangular');
    [num_non,den_non,bounds_non] = hinfid(h_noisy,1,'nonlinear',n,'triangular');

    % compute transfer functions
    H_lin = zeros(N,length(n));
    H_non = H_lin;
    for i=1:length(n),
        H_lin(:,i) = freqz(num_lin(i,:),den_lin(i,:),N,'whole');
        H_non(:,i) = freqz(num_non(i,:),den_non(i,:),N,'whole');
    end;

    % pulse response
    h_lin = zeros(N,length(n));
    h_non = h_lin;
    for i=1:length(n),
        notzeros = ~cumprod(all([num_lin(i,:);den_lin(i,:)]==0));
        h_lin(:,i) = dimpulse(num_lin(i,notzeros),den_lin(i,notzeros),N);
        notzeros = ~cumprod(all([num_non(i,:);den_non(i,:)]==0));
        h_non(:,i) = dimpulse(num_non(i,notzeros),den_non(i,notzeros),N);
    end;

    % error in response to random input (relative to exact RMS)
    random_input = randn(N,1);
    hr_lin = zeros(N,length(n));
    hr_non = hr_lin;
    for i=1:length(n),
        notzeros = ~cumprod(all([num_lin(i,:);den_lin(i,:)]==0));
        hr_lin(:,i) = dlsim(num_lin(i,notzeros),den_lin(i,notzeros),random_input);
        notzeros = ~cumprod(all([num_non(i,:);den_non(i,:)]==0));
        hr_non(:,i) = dlsim(num_non(i,notzeros),den_non(i,notzeros),random_input);
    end;
    hr_exact = dlsim(num,den,random_input);

    comp=0;
    save(datafile);
end;

% plot the pulse responses
clf('reset');
h=plot(t,h_noisy_stairs,'y-',t,h_exact_stairs,'w-');
set(h(1),'Color',g1+g2*.65,'LineWidth',1);
set(gca,'XLim',[0 50]);
```

hinfid_example1.m — Example I of H∞-based Identification (cont.)

```
sxlabel('\times time {\i k}'); sylabel('\times Pulse Response');
[hax,hli,hte] = slegend([.58 .65],h([2 1]),str2mat('\times\10System response', ...
                              '\times\10with noise, {\i \epsilon}=0.1'));
fn = 'hinfid_ex1_1_pulseresponse';
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
stitle(['\times Pulse Response of Original System and with Noise']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

% print out the models
for i=find(n(:)'<=10),
    notzeros = ~cumprod(all([num_lin(i,:);den_lin(i,:)]==0));
    disp(tf2str(num_lin(i,notzeros),den_lin(i,notzeros),'z',0, ...
                ['F_lin_' num2str(n(i)) '(z) = ']));
    notzeros = ~cumprod(all([num_non(i,:);den_non(i,:)]==0));
    disp(tf2str(num_non(i,notzeros),den_non(i,notzeros),'z',0, ...
                ['F_nonlin_' num2str(n(i)) '(z) = ']));
end;
bounds_non

% plot the transfer functions
clf('reset');
h=plot(omega/pi,abs(H_exact),omega/pi,abs(H_lin),omega/pi,abs(H_non));
l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
set(gca,'XLim',[0 1]);
for i=1:length(h), set(h(i),'LineStyle',deblank(linestyle(i,:)),'Color',graylevel(i,:), ...
                            'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
sylabel('\times Transfer Function Magnitude');
labs = str2mat('EXACT','{ }','Linear','Nonlinear','{ }','{\i n}=  5', ...
               '{\i n}=10','{\i n}=20','{\i n}=40','{\i n}=80');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
[hax,hli,hte] = slegend([.45 .7],[h(1);l;h(6);h(11);l;h(2:6)],labs);
fn = 'hinfid_ex1_2_transfunmag';
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
stitle(['\times Transfer Function of Exact System and Identified Models']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

clf('reset');
h=plot(omega/pi,angle(H_exact)/pi*180, ...
       omega/pi,angle(H_lin  )/pi*180, ...
       omega/pi,angle(H_non  )/pi*180);
l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
axis([0 1 -120 160]);
for i=1:length(h), set(h(i),'LineStyle',deblank(linestyle(i,:)),'Color',graylevel(i,:), ...
                            'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
sylabel('\times Transfer Function Phase [degrees]');
labs = str2mat('EXACT','{ }','Linear','Nonlinear','{ }','{\i n}=  5', ...
               '{\i n}=10','{\i n}=20','{\i n}=40','{\i n}=80');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
[hax,hli,hte] = slegend([.3 .7],[h(1);l;h(6);h(11);l;h(2:6)],labs);
fn = 'hinfid_ex1_3_transfunpha';
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
stitle(['\times Transfer Function of Exact System and Identified Models']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

% error in the transfer function
clf('reset');
h=plot(omega/pi,abs(H_exact*ones(1,length(n))-H_lin), ...
       omega/pi,abs(H_exact*ones(1,length(n))-H_non));
l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
set(gca,'XLim',[0 1]);
for i=2:length(h)+1, set(h(i-1),'LineStyle',deblank(linestyle(i,:)),'Color', ...
              graylevel(i,:),'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
sylabel('\times Magnitude of Transfer Function Error');
labs = str2mat('Linear','Nonlinear','{ }','{\i n}=  5', ...
               '{\i n}=10','{\i n}=20','{\i n}=40','{\i n}=80');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
[hax,hli,hte] = slegend([.3 .7],[h(5);h(10);l;h(1:5)],labs);
```

hinfid_example1.m — Example I of H∞-based Identification (cont.)

```
fn = 'hinfid_ex1_4_transfunerr';
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
stitle(['\times Transfer Function Error of Identified Models']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

delete(hax);
axis([0 1 0 .12]);
set(h([1 6]),'Visible','off');
[hax,hli,hte] = slegend([.23 .79],[h(5);h(10);1;h(2:5)],labs([1:3 5:8],:));
drawnow; if (scrn), pause; else,
fn = 'hinfid_ex1_5_transfunerr2';
printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']);
stitle('');
drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;

% pulse response
clf('reset');
h=plot(0:N-1,[h_exact h_lin h_non]);
l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
set(gca,'XLim',[0 20]);
for i=1:length(h), set(h(i),'LineStyle',deblank(linestyle(i,:)),'Color',graylevel(i,:), ...
                           'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
sxlabel('\times time {\i k}'); sylabel('\times Pulse Response');
labs = str2mat('EXACT','{ }','Linear','Nonlinear','{ }','{\i n}=  5', ...
               '{\i n}=10','{\i n}=20','{\i n}=40','{\i n}=80');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
[hax,hli,hte] = slegend([.45 .7],[h(1);l;h(6);h(11);l;h(2:6)],labs);
fn = 'hinfid_ex1_6_pulseresp';
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); end;
stitle(['\times Pulse Response of Exact System and Identified Models']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); end;

% absolute error in pulse response
clf('reset');
h=plot(0:N-1,abs(h_exact(:,ones(1,2*length(n)))-[h_lin h_non]));
l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
set(gca,'XLim',[0 20]);
for i=2:length(h)+1, set(h(i-1),'LineStyle',deblank(linestyle(i,:)),'Color', ...
              graylevel(i,:),'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
sxlabel('\times time {\i k}'); sylabel('\times Pulse Response Absolute Error');
labs = str2mat('Linear','Nonlinear','{ }','{\i n}=  5', ...
               '{\i n}=10','{\i n}=20','{\i n}=40','{\i n}=80');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
[hax,hli,hte] = slegend([.6 .55],[h(5);h(10);1;h(1:5)],labs);
fn = 'hinfid_ex1_7_pulseresperr';
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); end;
stitle(['\times Pulse Response Error of Identified Models']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); end;

% error in response to random input (relative to exact RMS)
clf('reset');
h=plot(0:N-1,abs(hr_exact(:,ones(1,2*length(n)))- ...
                 [hr_lin hr_non])/sqrt(sum(hr_exact.^2)/length(hr_exact)));
l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
set(gca,'XLim',[0 50]);
for i=2:length(h)+1, set(h(i-1),'LineStyle',deblank(linestyle(i,:)),'Color', ...
              graylevel(i,:),'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
sxlabel('\times time {\i k}'); sylabel('\times Relative Response Error');
labs = str2mat('Linear','Nonlinear','{ }','{\i n}=  5', ...
               '{\i n}=10','{\i n}=20','{\i n}=40','{\i n}=80');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
[hax,hli,hte] = slegend([.8 .77],[h(5);h(10);1;h(1:5)],labs);
fn = 'hinfid_ex1_8_randresperr';
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); end;
stitle(['\times Response Error (relative to exact RMS)' ...
        ' of Identified Models with Random Input']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); end;

% relative difference between linear and nonlinear response to random input
clf('reset');
```

hinfid_example1.m — Example I of H∞-based Identification (cont.)

```
h=plot(0:N-1,abs((hr_lin-hr_non)*2./(hr_lin+hr_non)));
set(gca,'XLim',[0 50]);
for i=2:length(h)+1, set(h(i-1),'LineStyle',deblank(linestyle(i,:)),'Color', ...
              graylevel(i,:),'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
set(h(1),'LineStyle',deblank(linestyle(11,:)),'Color',graylevel(11,:), ...
         'LineWidth',linewidth(7),'MarkerSize',markersize(11));
sxlabel('\times time {\i k}'); sylabel('\times Relative Response Difference');
labs = str2mat('{\i n}=  5','{\i n}=10','{\i n}=20','{\i n}=40','{\i n}=80');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
[hax,hli,hte] = slegend([.3 .7],h(1:5),labs);
fn = 'hinfid_ex1_9_randrespdiff';
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); end;
stitle(['\times Relative Response Difference between Linear and ' ...
        'Nonlineear Identified Models with Random Input']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); end;
```

8.2.4 `hinfid_example2.m` — Example II of H_∞-based Identification

`hinfid_example2` runs the second example of H_∞-based identification. Its task is the identification of a six degree of freedom system with single input and six outputs.

```
% hinfid_example2.m
echo on

% This runs the second H-infinity identification example
%
% The system is a 6DOF train with identical masses, springs, and dashpots.


% load in past data if already run
datafile = 'hinfid_ex2';
eval('load(datafile);','comp=1;');


% set up some variables
scrn = 1;  % change to 0 to do hardcopies

if (scrn),
    g1 = 1;
    g2 = -[1 1 1];
else,
    g1 = 0;
    g2 = [1 1 1];
end;


% set line colors, styles, and widths
linestyle=str2mat('x',':','-.','--','-'); linestyle=str2mat('-',linestyle,linestyle);
linewidth=  [16;   4 ; 7 ; 1 ; 1 ; 1 ;     6 ; 2 ; 8 ; 8 ; 8 ]/8;
markersize= [ 6;  2.5; 6 ; 6 ; 6 ; 6 ;    2.5; 6 ; 6 ; 6 ; 6 ];
graylevel=1-[ 1;   1 ; 1 ; 1 ; 1 ; 1 ;     .4; 1 ; .4; .4; .4];
mask=(graylevel==0|graylevel==1|scrn);
graylevel(mask) = (1-graylevel(mask));
graylevel = graylevel(:,[1 1 1]);



%%%%%%%%%%%%%%%%
%              %
%  Example II  %
%              %
%%%%%%%%%%%%%%%%

if (comp),
    % some parameters
    T = .8;                  % sampling time
    nn = 6;                  % # of degrees of freedom in real system
    input_number = 1;        % because we can only do SIMO H-inf ID problems
    N = 2048;                % number of points in pulse response
    n = 2*nn*[2 5 10 20 40]; % model orders
    window = 'trapezoidal';  % window to use in identification
    epsilon = 0.1;           % noise rms relative to pulse response rms

    % the exact system specifications
    [A,B,C,D,MM,CC,KK,PP,L,W,Z] = ndof(6,T);
    B = B(:,input_number);
    D = D(:,input_number);

    % simulate the system
    h_exact = dimpulse(A,B,C,D,1,N);

    % create the noise
    rand('seed',21217); % so we can repeat this exactly
    Noise(1:N/2+1,1:nn) = [zeros(1,nn); ...
                           exp(sqrt(-1)*rand(N/2-1,nn)*2*pi); ...
                           sign(rand(1,nn)-.5)];
```

hinfid_example2.m — Example II of H∞-based Identification (cont.)

```
    Noise(N/2+2:N,1:nn) = conj(Noise(N/2:-1:2,1:nn));
    noise = real(ifft(Noise));
    h_noisy = h_exact + noise*epsilon*diag(sqrt(sum(h_exact.^2)./sum(noise.^2)));

    % do the identification
    bounds_non=zeros(size(n));
    for i=1:length(n), % loop to save memory
        if (n(i)<150), roots_tolerance=[]; else, roots_tolerance=-1; end;
        [num_lin1,den_lin1]                = hinfid(h_noisy,nn,'linear'   , ...
                                                    n(i),window,[],roots_tolerance);
        [num_non1,den_non1,bounds_non(i)] = hinfid(h_noisy,nn,'nonlinear', ...
                                                    n(i),window,[],roots_tolerance);
        if (i==1),
            num_lin = num_lin1;
            den_lin = den_lin1;
            num_non = num_non1;
            den_non = den_non1;
        else,
            nz = size(num_lin1,2) - size(num_lin,2);
            num_lin = [zeros(size(num_lin,1),nz) num_lin; ...
                       zeros(size(num_lin1,1),-nz) num_lin1];
            nz = size(den_lin1,2) - size(den_lin,2);
            den_lin = [zeros(size(den_lin,1),nz) den_lin; ...
                       zeros(size(den_lin1,1),-nz) den_lin1];
            nz = size(num_non1,2) - size(num_non,2);
            num_non = [zeros(size(num_non,1),nz) num_non; ...
                       zeros(size(num_non1,1),-nz) num_non1];
            nz = size(den_non1,2) - size(den_non,2);
            den_non = [zeros(size(den_non,1),nz) den_non; ...
                       zeros(size(den_non1,1),-nz) den_non1];
        end;
    end;

    comp=0;
    save(datafile);
end;
bounds_non

% plot pulse response of one output
output_number = 3;
clf('reset');
h=plot(0:N-1,[h_noisy(:,output_number) h_exact(:,output_number)],'w-');
set(h(1),'Color',g1+g2*.65,'LineWidth',1);
axis([0 N-1 [-1 1]*max(abs(get(gca,'YLim')))]);
sxlabel('\times time {\i k}'); sylabel('\times Pulse Response');
[hax,hli,hte] = slegend([.7 .7],h([2 1]),str2mat('\times\10System response', ...
                                                 '\times\10with noise, {\i \epsilon}=0.1'));
fn = 'hinfid_ex2_1_pulseresponse';
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
stitle(['\times Pulse Response of Mass #' num2str(output_number) ...
        ' of Original System and with Noise']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

% compute transfer functions
NN = 4096;
H_lin = zeros(NN,nn*length(n));
H_non = H_lin;
for i=1:nn*length(n),
    H_lin(:,i) = freqz(num_lin(i,:),den_lin(i,:),NN);
    H_non(:,i) = freqz(num_non(i,:),den_non(i,:),NN);
end;
[num_exact,den_exact] = ss2tf(A,B,C,D,1);
H_exact = zeros(NN,nn);
for i=1:nn,
    [H_exact(:,i),omega] = freqz(num_exact(i,:),den_exact,NN);
end;

% plot the exact transfer functions and the modal transfer functions
clf('reset');
ww=[1;1]*sort(W(:)'*T/pi); ww=ww(:); mm=1:length(ww); mm=10.^(200*[(rem(mm,4)-1>0)-.5]);
```

hinfid_example2.m — Example II of H∞-based Identification (cont.)

```
subplot(2,1,1); h=semilogy(ww,mm,'y--',omega/pi,abs(H_exact),'w-');
set(h(1),'LineWidth',1,'Color',.4*[1 1 1]); set(h(2:length(h)),'LineWidth',.125);
axis([0 .6 .02 100]);
set(gca,'XTickLabels','');
sylabel('\times Transfer Function Magnitude');
subplot(2,1,2); h=semilogy(ww,mm,'y--',omega/pi,abs(H_exact*PP.'),'w-');
set(h(1),'LineWidth',1,'Color',.4*[1 1 1]); set(h(2:length(h)),'LineWidth',.125);
axis([0 .6 .02 100]);
sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
sylabel('\times Modal Transfer Function Magnitude');
subplot(2,1,1);
fn = 'hinfid_ex2_2_tfmag_exact';
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
stitle('\times Transfer Function of Exact System and to Modal Displacements');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

% plot some of the transfer functions
nni=1:5;
for output_number=[1 3];
    clf('reset');
    h=semilogy(omega/pi,abs(H_exact(:,output_number)),          ...
               omega/pi,abs(H_lin(:,(nni-1)*6+output_number)), ...
               omega/pi,abs(H_non(:,(nni-1)*6+output_number)));
    axis([0 .6+10*eps .01 40]);
    l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
    for i=1:length(h), set(h(i),'LineStyle',deblank(linestyle(i,:)),'Color',graylevel(i,:), ...
                               'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
    sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
    sylabel('\times Transfer Function Magnitude');
    labs = str2mat('EXACT','{ }','Linear','Nonlinear','{ }','{\i n}=  24', ...
                   '{\i n}=  60','{\i n}=120','{\i n}=240','{\i n}=480');
    labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
    [hax,hli,hte] = slegend([.81 .76],[h(1);l;h(6);h(11);l;h(2:6)],labs);
    fn = ['hinfid_ex2_3_tfmags' num2str(output_number)];
    if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
    stitle(['\times Transfer Function #' num2str(output_number) ...
            ' of Exact System and Identified Models']);
    drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

    clf('reset');
    h=semilogy(omega/pi,abs(H_exact(:,output_number)),          ...
               omega/pi,abs(H_lin(:,(nni-1)*6+output_number)));
    axis([0 .6+10*eps .01 40]);
    l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
    for i=1:length(h), set(h(i),'LineStyle',deblank(linestyle(i,:)),'Color',graylevel(i,:), ...
                               'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
    set(h(2),'LineStyle',deblank(linestyle(11,:)),'Color',graylevel(11,:), ...
             'LineWidth',linewidth(7),'MarkerSize',markersize(11));
    sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
    sylabel('\times Transfer Function Magnitude');
    labs = str2mat('EXACT','{ }','{\i n}=  24', ...
                   '{\i n}=  60','{\i n}=120','{\i n}=240','{\i n}=480');
    labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
    [hax,hli,hte] = slegend([.81 .76],[h(1);l;h(2:6)],labs);
    fn = ['hinfid_ex2_31_tfmags' num2str(output_number)];
    if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
    stitle(['\times Transfer Function #' num2str(output_number) ...
            ' of Exact System and Linear Identified Models']);
    drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

    clf('reset');
    h=semilogy(omega/pi,abs(H_exact(:,output_number)),          ...
               omega/pi,abs(H_non(:,(nni-1)*6+output_number)));
    axis([0 .6+10*eps .01 40]);
    l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
    for i=1:length(h), set(h(i),'LineStyle',deblank(linestyle(i,:)),'Color',graylevel(i,:), ...
                               'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
    set(h(2),'LineStyle',deblank(linestyle(11,:)),'Color',graylevel(11,:), ...
             'LineWidth',linewidth(7),'MarkerSize',markersize(11));
    sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
```

hinfid_example2.m — Example II of H∞-based Identification (cont.)

```
sylabel('\times Transfer Function Magnitude');
labs = str2mat('EXACT','{ }','{\i n}=  24', ...
               '{\i n}=  60','{\i n}=120','{\i n}=240','{\i n}=480');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
[hax,hli,hte] = slegend([.81 .79],[h(1);1;h(2:6)],labs);
fn = ['hinfid_ex2_3n_tfmags' num2str(output_number)];
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
stitle(['\times Transfer Function #' num2str(output_number) ...
        ' of Exact System and Nonlinear Identified Models']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

clf('reset');
h=plot(omega/pi,180/pi*angle(H_exact(:,output_number)),          ...
       omega/pi,180/pi*angle(H_lin(:,(nni-1)*6+output_number)), ...
       omega/pi,180/pi*angle(H_non(:,(nni-1)*6+output_number)));
set(gca,'XLim',[0 .6+10*eps]);
l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
for i=1:length(h), set(h(i),'LineStyle',deblank(linestyle(i,:)),'Color',graylevel(i,:), ...
                           'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
sylabel('\times Transfer Function Phase [degrees]');
labs = str2mat('EXACT','{ }','Linear','Nonlinear','{ }','{\i n}=  24', ...
               '{\i n}=  60','{\i n}=120','{\i n}=240','{\i n}=480');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
if (output_number==1), legpos=[.58 .7]; else, legpos=[.53 .3]; end;
[hax,hli,hte] = slegend(legpos,[h(1);1;h(6);h(11);1;h(2:6)],labs);
fn = ['hinfid_ex2_4_tfphas' num2str(output_number)];
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
stitle(['\times Transfer Function #' num2str(output_number) ...
        ' of Exact System and Identified Models']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

clf('reset');
h=plot(omega/pi,180/pi*angle(H_exact(:,output_number)),          ...
       omega/pi,180/pi*angle(H_lin(:,(nni-1)*6+output_number)));
set(gca,'XLim',[0 .6+10*eps]);
l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
for i=1:length(h), set(h(i),'LineStyle',deblank(linestyle(i,:)),'Color',graylevel(i,:), ...
                           'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
set(h(2),'LineStyle',deblank(linestyle(11,:)),'Color',graylevel(11,:), ...
         'LineWidth',linewidth(7),'MarkerSize',markersize(11));
sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
sylabel('\times Transfer Function Phase [degrees]');
labs = str2mat('EXACT','{ }','{\i n}=  24', ...
               '{\i n}=  60','{\i n}=120','{\i n}=240','{\i n}=480');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
if (output_number==1), legpos=[.58 .7]; else, legpos=[.53 .3]; end;
[hax,hli,hte] = slegend(legpos,[h(1);1;h(2:6)],labs);
fn = ['hinfid_ex2_4l_tfphas' num2str(output_number)];
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
stitle(['\times Transfer Function #' num2str(output_number) ...
        ' of Exact System and Linear Identified Models']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

clf('reset');
h=plot(omega/pi,180/pi*angle(H_exact(:,output_number)),          ...
       omega/pi,180/pi*angle(H_non(:,(nni-1)*6+output_number)));
set(gca,'XLim',[0 .6+10*eps]);
l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
for i=1:length(h), set(h(i),'LineStyle',deblank(linestyle(i,:)),'Color',graylevel(i,:), ...
                           'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
set(h(2),'LineStyle',deblank(linestyle(11,:)),'Color',graylevel(11,:), ...
         'LineWidth',linewidth(7),'MarkerSize',markersize(11));
sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
sylabel('\times Transfer Function Phase [degrees]');
labs = str2mat('EXACT','{ }','{\i n}=  24', ...
               '{\i n}=  60','{\i n}=120','{\i n}=240','{\i n}=480');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
if (output_number==1), legpos=[.58 .7]; else, legpos=[.53 .3]; end;
[hax,hli,hte] = slegend(legpos,[h(1);1;h(2:6)],labs);
```

hinfid_example2.m — Example II of H∞-based Identification (cont.)

```
fn = ['hinfid_ex2_4n_tfphas' num2str(output_number)];
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
stitle(['\times Transfer Function #' num2str(output_number) ...
        ' of Exact System and Nonlinear Identified Models']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

clf('reset');
h=semilogy(omega/pi,abs(H_exact(:,output_number)*ones(size(nni))- ...
                        H_lin(:,(nni-1)*6+output_number)), ...
           omega/pi,abs(H_exact(:,output_number)*ones(size(nni))- ...
                        H_non(:,(nni-1)*6+output_number)));
axis([0 .6+10*eps 5e-5 20])
l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
for i=2:length(h)+1, set(h(i-1),'LineStyle',deblank(linestyle(i,:)),'Color', ...
                graylevel(i,:),'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
sylabel('\times Transfer Function Error Magnitude');
labs = str2mat('Linear','Nonlinear','{ }','{\i n}=  24', ...
               '{\i n}=  60','{\i n}=120','{\i n}=240','{\i n}=480');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
[hax,hli,hte] = slegend([.81 .79],[h(5);h(10);1;h(1:5)],labs);
fn = ['hinfid_ex2_5_tfmagserr' num2str(output_number)];
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
stitle(['\times Transfer Function #' num2str(output_number) ...
        ' Error of Identified Models']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

clf('reset');
h=semilogy(omega/pi,abs(H_exact(:,output_number)*ones(size(nni))- ...
                        H_lin(:,(nni-1)*6+output_number)));
axis([0 .6+10*eps 5e-5 20])
l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
for i=2:length(h)+1, set(h(i-1),'LineStyle',deblank(linestyle(i,:)),'Color', ...
                graylevel(i,:),'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
set(h(1),'LineStyle',deblank(linestyle(11,:)),'Color',graylevel(11,:), ...
         'LineWidth',linewidth(7),'MarkerSize',markersize(11));
sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
sylabel('\times Transfer Function Error Magnitude');
labs = str2mat('{\i n}=  24', ...
               '{\i n}=  60','{\i n}=120','{\i n}=240','{\i n}=480');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
[hax,hli,hte] = slegend([.81 .81],[h(1:5)],labs);
fn = ['hinfid_ex2_5l_tfmagserr' num2str(output_number)];
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
stitle(['\times Transfer Function #' num2str(output_number) ...
        ' Error of Linear Identified Models']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

clf('reset');
h=semilogy(omega/pi,abs(H_exact(:,output_number)*ones(size(nni))- ...
                        H_non(:,(nni-1)*6+output_number)));
axis([0 .6+10*eps 5e-5 20])
l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
for i=2:length(h)+1, set(h(i-1),'LineStyle',deblank(linestyle(i,:)),'Color', ...
                graylevel(i,:),'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
set(h(1),'LineStyle',deblank(linestyle(11,:)),'Color',graylevel(11,:), ...
         'LineWidth',linewidth(7),'MarkerSize',markersize(11));
sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
sylabel('\times Transfer Function Error Magnitude');
labs = str2mat('{\i n}=  24', ...
               '{\i n}=  60','{\i n}=120','{\i n}=240','{\i n}=480');
labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
[hax,hli,hte] = slegend([.81 .81],[h(1:5)],labs);
fn = ['hinfid_ex2_5n_tfmagserr' num2str(output_number)];
if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
stitle(['\times Transfer Function #' num2str(output_number) ...
        ' Error of Nonlinear Identified Models']);
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

clf('reset');
```

hinfid_example2.m — Example II of H∞-based Identification (cont.)

```
    h=semilogy(omega/pi,abs(H_lin(:,(nni-1)*6+output_number)- ...
                           H_non(:,(nni-1)*6+output_number)));
    axis([0 .6+10*eps .0003 .2])
    l=line(.1,.1,'Color','k'); set(l,'XData',[],'YData',[],'ZData',[]);
    for i=2:length(h)+1, set(h(i-1),'LineStyle',deblank(linestyle(i,:)),'Color', ...
                  graylevel(i,:),'LineWidth',linewidth(i),'MarkerSize',markersize(i)); end;
    set(h(1),'LineStyle',deblank(linestyle(11,:)),'Color',graylevel(11,:), ...
             'LineWidth',linewidth(7),'MarkerSize',markersize(11));
    sxlabel('\times normalized frequency {\i\omega T}/{\i\pi}');
    sylabel('\times Transfer Function Difference Magnitude');
    labs = str2mat('{\i n}=  24', ...
                   '{\i n}=  60','{\i n}=120','{\i n}=240','{\i n}=480');
    labs = setstr([ones(size(labs,1),1)*'\times\10' labs]);
    [hax,hli,hte] = slegend([.4 .23],[h(1:5)],labs);
    fn = ['hinfid_ex2_6_tfmagsdiff' num2str(output_number)];
    if (~scrn), drawnow; printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
    stitle(['\times Magnitude of Difference between Linear and Nonlinear' ...
            ' Identified Models (#' num2str(output_number) ')']);
    drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

end;
```

8.2.5 `tf2str.m` — Convert Transfer Function to String

`tf2str` converts a transfer function, specified by numerator and denominator polynomial coefficient row vectors, into a printable string.

```
function out = tf2str(num,den,s,fact,initstr)
% TF2STR  Converts transfer function to printable string.
%
%   TF2STR(NUM,DEN,S,FACT,INITSTR) converts a transfer function to a
%                                  printable string representation.
%
%   TF2STR(NUM,DEN) is the default.  Other arguments are optional.
%
%          S is the character used for the variable, default 's'.
%
%          FACT will cause the numerator and denominator to be printed
%          in a factored representation (unless FACT=0 or 'n');
%
%          INITSTR is an initial string to be printed, such as 'X(s) = '.
%
%          Note that NUM must be SISO (i.e., single row vector).
%
%   See also POLY2STR, POLY2TEXT.

% Copyright (c) 1993, Erik A. Johnson <johnsone@uxh.cso.uiuc.edu>, 11/22/93.

% check arguments
if nargin==5,
    if ~isstr(initstr),
        error('INITSTR must be a string.');
    end;
else,
    if nargin<5,
        initstr = '';
    end;
    if nargin<4,
        fact=0;
    elseif (isstr(fact)),
        if (fact=='n'),
            fact = 0;
        else,
            fact = 1;
        end;
    else,
        fact = (fact ~= 0);
    end;
    if nargin<3,
        s = 's';
    end;
    if nargin<2,
        den=[1];
    end;
    if nargin<1,
        error('Got to have some arguments!');
    end;
end;

[nr,nc] = size(num);
if (nr>1 & nc>1),
    error('NUM must be SISO (i.e., a single row vector).');
end;

[dr,dc] = size(den);
if (dr*dc==1),
    out = [initstr poly2text(num/den,s,fact)];
else,
    if (fact),
        while (length(num)>1 & num(1)==0),
            num = num(2:length(num));
        end;
        while (length(den)>1 & den(1)==0),
```

tf2str.m — Convert Transfer Function to String (cont.)

```
            den = den(2:length(den));
        end;
        if (den(1)~=0),
            coef = num(1)/den(1);
            if (num(1)~=0),
                num = num/num(1);
            end;
            den = den/den(1);
        else
            coef = 1;
        end;
        if (coef == -1),
            initstr = [initstr '- '];
        elseif (coef ~= 1),
            initstr = [initstr num2str(coef) ' '];
        end;
    end;
    numstr = poly2text(num,s,fact);
    denstr = poly2text(den,s,fact);
    len = max(length(numstr),length(denstr))+2;
    numstr=[blanks(floor((len-length(numstr))/2)) numstr blanks(ceil((len-length(numstr))/2))];
    denstr=[blanks(floor((len-length(denstr))/2)) denstr blanks(ceil((len-length(denstr))/2))];
    out = [blanks(length(initstr)) numstr; initstr strrep(blanks(len),' ','-');
                blanks(length(initstr)) denstr];
end;
```

8.2.6 `poly2text.m` — Convert Polynomial to String

`poly2text` converts a row vector of polynomial coefficients to a printable string representation.

```
function out = poly2text(p,s,fact)
% POLY2TEXT  Printable string representation of a polynomial.
%
%   POLY2TEXT(P,S,FACT) converts a polynomial to a printable string
%                       representation.  S is the character to be used
%                       for the variable (default is 's').  FACT (if
%                       given and (FACT~=0)&(FACT~='n')) returns the
%                       polynomial in a factored representation.
%
%   This function differs from POLY2STR in that it does the factorization
%   and it doesn't print leading blanks.
%
%   See also POLY2STR in the Control System Toolbox.

% Copyright (c) 1993, Erik A. Johnson <johnsone@uxh.cso.uiuc.edu>, 11/22/93.

% check arguments
if nargin<3,
    fact=0;
elseif (isstr(fact)),
        if (fact=='n'),
            fact = 0;
        else,
            fact = 1;
        end;
    end;
else,
    fact = (fact ~= 0);
end;
if nargin<2,
    s = 's';
end;
if nargin<1,
    error('Got to have some arguments!');
end;

% do the work
if (length(p)==1 | ~fact),
    out = poly2str(p,s);
    while (length(out)>1 & out(1)==' '),
        out = out(2:length(out));
    end;
    out = deblank(out);
else,
    while (length(p)>1 & p(1)==0),
        p = p(2:length(p));
    end;
    if (p(1) == -1),
        out = '-';
        p = -p;
    elseif (p(1)~=1 & p(1)~=0),
        out = num2str(p(1));
        p = p / p(1);
    else,
        out = '';
    end;
    rr = roots(p);
    i = 1;
    while (i<=length(rr)),
        if (imag(rr(i)) ~= 0),
            rstr = poly2str(real(poly([rr(i) rr(i+1)])),s);
            i = i + 1;
        else
            rstr = poly2str([1 -rr(i)],s);
        end;
        while (length(rstr)>1 & rstr(1)==' '),
```

poly2text.m — Convert Polynomial to String (cont.)

```
            rstr = rstr(2:length(rstr));
        end;
        rstr = deblank(rstr);
        out = [out ' (' rstr ')'];
        i = i + 1;
    end;
    if (out(1)==' '),
        out = out(2:length(out));
    end;
end;
```

8.3 EIGENSYSTEM REALIZATION ALGORITHM CODES

8.3.1 `era.m` — Eigensystem Realization Algorithm

era is an implementation of the Eigensystem Realization Algorithm developed by J.-N. Juang and colleagues.

```
function [A,B,C,poles,mshapes,wn,zn,MAC,nout] = era(Y,T,ni,n,rs,k)
% ERA  Identification via the Eigensystem Realization Algorithm.
%
%   [A,B,C] = ERA(Y,T,NI,N,RS) uses the Eigensystem Realization Algorithm
%                              (ERA) to identify the system with a given
%         pulse response.  The input and output arguments are:
%
%            Y     is the pulse response (each column is a pulse response;
%                  the response of the i-th output to a pulse on the j-th
%                  input is in column (i+(j-1)*no) of Y, where no is the
%                  number of outputs)
%
%            T     is the sampling period of the system (defaults to 1)
%
%            NI    is the number of inputs (defaults to 1)
%
%            N     is a vector of candidate system orders.  Alternately,
%                  candidate singular value cutoff tolerances (i.e., a
%                  level below which singular values are considered zero)
%                  can be passed in N.  If all elements of N are positive
%                  integers, the former is assumed, otherwise the latter.
%                  A further possible value, N='prompt' or N='ask' will
%                  plot the singular values and ask for a cut-off
%                  tolerance or order (clicking in the graph window will
%                  display the point at which the mouse is clicked).
%
%            RS    takes the form of [R S] and is the number of row blocks
%                  R and column blocks S should be used in forming the
%                  generalized Hankel matrices.  They should generally
%                  both be at least twice N.  If not supplied, they are
%                  chosen to be floor((size(Y,1)-K-1)/2) to use as much of
%                  the pulse response Y as possible; this may often be far
%                  too big (causing a huge, but unnecessary, increase in
%                  computation time).  If RS is a column vector of the
%                  same length as N or is a scalar, then R and S are chosen
%                  to be RS*N (this requires N to be explicitly specified,
%                  and not 'ask').
%
%            A,B,C are the discrete-time state-space matrices of the
%                  identified model.  If N is a non-scalar vector, then
%                  the models are given by A=[A1;A2;...], B=[B1;B2;...];
%                  C=[C1 C2 ...], where the Ai will be padded with extra
%                  columns if the values of N are not all the same.
%
%   [A,B,C,POLES,MSHAPES,WN,ZN,MAC,N] = ERA(...) also returns the complex
%                                                 discrete-time poles and
%         modeshapes, continuous-time natural frequencies (in rads/sec)
%         and damping ratios, the Modal Amplitude Coherence (MAC), and the
%         actual system orders, respectively.  (The MAC is a measure of
%         whether a mode is "true" or noise- induced; it is always in [0,1],
%         with smaller values signifying noise-induced modes.)  For non-
%         scalar N, the poles,  natural frequencies, damping ratios, and
%         MACs for each candidate model are stacked like the input matrix B
%         above and the modeshapes are stacked like the state matrix A above.
%
%         NOTE:  The algorithm to compute the values of MAC has not been
%                fully tested and may not give accurate MAC results.
%
%   ... = ERA(...,K) also gives the time offset used to compute the model.
%                    Its default value is 1.

% Copyright (c)1996, Erik A. Johnson <johnsone@uiuc.edu>, 5/30/96
```

era.m — Eigensystem Realization Algorithm (cont.)

```
% Some ideas for this code were taken from another implementation of ERA
% in MATLAB by Allen Prell, graduate student, Dept. of Aero & Astro Engrg.,
% U. of Illinois, December 1994.

% The primary source for the ERA algorithm used here is:
%
%    J.-N. Juang and R.S. Pappa, 1985.  "An Eigensystem Realization Algorithm
%    for Modal Parameter Identification and Model Reduction."  Journal of
%    Guidance, Control, and Dynamics, 8(5), Sept.-Oct. 1985, 620-627.

% check # of arguments
if (nargin<1), error('ERA requires at least 1 input argument.');
elseif (nargin>6), error('ERA takes at most 6 input arguments.');
elseif (nargout>9), error('ERA produces at most 9 outputs.');
end;

% handle callback
if isstr(Y),
    strformat = 'N=%.0f,  tol=%g';
    linetag = 'pt line';
    txt1tag = 'pt text';
    txt2tag = 'pt text2';
    l  = findobj(gcf,'Tag',linetag);
    t1 = findobj(gcf,'Tag',txt1tag);
    t2 = findobj(gcf,'Tag',txt2tag);
    ptfig = get(gcf,'CurrentPoint');
    posfig = get(gcf,'Position');
    pt = get(gca,'CurrentPoint');
    pt = pt(1,1:2);
    axlims = axis;
    str = sprintf(strformat,pt);
    if strcmp(Y,'ButtonDown'),
        if (0), % text method
            disp(str);
        else,
            delete([l;t1;t2]);
            if all(pt>=axlims([1 3]) & pt<=axlims([2 4])),
                erasemode = 'xor';
                l = line(pt(1),pt(2),'LineStyle','+','Color','w', ...
                         'EraseMode',erasemode,'Tag',linetag);
                t = text([1;1]*pt(1),[1;1]*pt(2),str([1;1],:), ...
                         'EraseMode',erasemode,'Visible','off', ...
                         'FontSize',round(get(gca,'DefaultTextFontSize')*.83));
                set(t(1),'Tag',txt1tag);
                set(t(2),'Tag',txt2tag);
                set(gcf,'WindowButtonUpFcn','era(''ButtonUp'');', ...
                        'WindowButtonMotionFcn','era(''MouseMoved'');');
                era('MouseMoved');
            end;
        end;
    elseif strcmp(Y,'MouseMoved') & all(ptfig>=0&ptfig<=posfig(3:4)),
        if all(pt>=axlims([1 3]) & pt<=axlims([2 4])),
            if ~isempty(l), set(l,'Xdata',pt(1),'Ydata',pt(2)); end;
            if ~isempty(t2),
                set(t2,'String',str,'Position',[pt 0],'HorizontalAlignment','left');
                moveit = get(t2,'FontSize');
                set(t2,'Units','pixels');
                set(t2,'Position',get(t2,'Position')+[moveit 0]);
                set(t2,'Units','data');
                ext = get(t2,'Extent');
                if (ext(1)+ext(3) >= axlims(2)),
                    set(t2,'Position',[pt 0],'HorizontalAlignment','right');
                    set(t2,'Units','pixels');
                    set(t2,'Position',get(t2,'Position')-[moveit 0]);
                    set(t2,'Units','data');
                end;
                if ~isempty(t1),
                    set(t1,'Visible','off','Position',get(t2,'Position'), ...
                           'HorizontalAlignment',get(t2,'HorizontalAlignment'), ...
                           'String',str,'Visible','on');
```

era.m — Eigensystem Realization Algorithm (cont.)

```
                end;
            end;
        end;
    else,
        delete([1;t1;t2]);
        set(gcf,'WindowButtonUpFcn','1;','WindowButtonMotionFcn','');
        if strcmp(Y,'ButtonUp'), disp(str); end;
    end;
    return;
end;

% handle unsupplied arguments
if (nargin<2), T=[];  end;
if (nargin<3), ni=[]; end;
if (nargin<4), n=[];  end;
if (nargin<5), rs=[]; end;
if (nargin<6), k=[];  end;

% determine some sizes
nk = size(Y,1);
no = size(Y,2)/ni;
if (no~=round(no)),
    error('ERA requires the # of columns of Y to be a multiple of NI.');
end;

% set default values
if isempty(T),  T=1;     else, T=T(1);       end;
if isempty(ni), ni=1;    else, ni=ni(1);     end;
if isempty(n),  n='ask';                     end;
if isempty(rs), rs=[1 1]*floor((nk-k-1)/2); end;
if isempty(k),  k=1;     else, k=k(1);       end;

% check size of rs
if (size(rs,2)>2),
    if (size(rs,1)<=2),
        rs=rs.';
    else,
        error('ERA requires that RS have at most 2 columsn.');
    end;
end;

% determine how many candidates we have
if isstr(n), nn=0; else, n=n(:).'; nn=length(n); end;
rs=rs.'; nrs=size(rs,2);
if all([nn nrs]<=1),
    nmodels = 1;
elseif (all([nn nrs]>1) & nn~=nrs),
    error('ERA requires that N and RS be empty, have one row, or have the same # of rows.');
else,
    nmodels = max([nn nrs]);
end;
if (nn==1  & nmodels>1), n =n( :,ones(1,nmodels)); end;
if (nrs==1 & nmodels>1), rs=rs(:,ones(1,nmodels)); end;
if (size(rs,1)==1), if (nn>0), rs=n.*rs; end; rs=rs([1;1],:); end;
nistol = ~isstr(n) & any(n~=round(n)|n<.5);

% if n is known, preallocate the outputs
if ~(isstr(n) | nistol),
    maxn = max(n);
    sumn = sum(n);
    A = zeros(sumn,maxn);
    if (nargout>=2), B=zeros(sumn,ni);       end;
    if (nargout>=3), C=zeros(no,sumn);       end;
    if (nargout>=4), poles=zeros(sumn,1);    end;
    if (nargout>=5), mshapes=zeros(sumn,1); end;
    if (nargout>=8), MAC=zeros(sumn,1);      end;
end;
nout=zeros(nmodels,1);

% construct the huge Hankel matrix
```

era.m — Eigensystem Realization Algorithm (cont.)

```
maxrs = [max(rs(1,:)) max(rs(2,:))+k]+1;
H = zeros([no ni].*maxrs);
maxnk = max(sum(rs))+1+k;
if (nk<maxnk),
    error('ERA did not have enough times in Y for the requested [R S].');
end;
Y2 = reshape(Y(1:maxnk,:).',no,ni*maxnk);
for j=1:maxrs(1),
    Y2cols = (j-1)*ni+1:min(ni*maxnk,(maxrs(2)+j-1)*ni);
    H((j-1)*no+1:j*no,1:length(Y2cols)) = Y2(:,Y2cols);
end;

% loop over the candidate model orders
for j=1:nmodels,
    % do the singular value decomposition
    r = rs(1,j);
    s = rs(2,j);
    [P,D,Q] = svd(H(1:no*(r+1),1:ni*(s+1)),0);
    sv = diag(D);
    % determine the truncation point
    if isstr(n),
        % graph and get the value in ncur
        h=semilogy(sv(:,[1 1])); xlabel('N'); ylabel('Singular Values');
        axis(axis);
        set(h(2),'LineStyle','o');
        title(['Singular values for [R S] = ' mat2str(rs(:,j).')]);
        set(gcf,'WindowButtonDownFcn','era(''ButtonDown'');', ...
                'WindowButtonUpFcn','1;');
        drawnow;
        ncur = input('Enter order or cutoff tolerance:  ');
        set(gcf,'WindowButtonDownFcn','');
        title(''); drawnow;
        if isempty(ncur), ncur=length(sv); end;
        nistol = any(ncur~=round(ncur)|ncur<.5);
    else,
        ncur = n(j);
    end;
    if (nistol),
        ncur = sum(sv>=ncur);
    end;
    % do the truncation
    D_half = diag(sqrt(sv(1:ncur)));
    D_invhalf = diag(sqrt(1./sv(1:ncur)));
    P(:,ncur+1:size(P,2)) = [];
    Q(:,ncur+1:size(Q,2)) = [];
    % compute the state-space system matrices
    Ai = D_invhalf * P' * H(1:no*(r+1),ni*k+1:ni*(s+1+k)) * Q * D_invhalf;
    Bi = D_half * Q' * [eye(ni);zeros(s*ni,ni)];
    Ci = [eye(no) zeros(no,r*no)] * P * D_half;
    % insert into outputs
    oldn = sum(nout);
    nout(j,1) = ncur;
    irows = oldn+1:oldn+ncur;
    A(irows,1:ncur) = Ai;
    B(irows,1:ni) = Bi;
    C(1:no,irows) = Ci;
    % compute poles and modeshapes
    if (nargout>=4),
        if (nargout<5),
            polesi = eig(Ai);
        else,
            [mi,polesi] = eig(Ai);
            polesi = diag(polesi);
            mshapes(irows,1:nout(j)) = mi;
        end;
        poles(irows,1) = polesi;
    end;
    % compute modal amplitude coherence
    if (nargout>=8),
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

era.m — Eigensystem Realization Algorithm (cont.)

```
        % NOTE: This method of computing the modal amplitude coherence %
        %       (MAC) values has not been fully tested and its results %
        %       may or may not be accurate.                            %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        g = Q * D_invhalf * inv(mi)';
        b = [eye(ni) zeros(ni,s*ni)] * g;
        ii = (1:ni).'; ii=ii(:,ones(1,s+1)); ii=ii(:);
        jj = (0:s).'; jj=jj(:,ones(1,ncur));
        zz = polesi(:,ones(1,ni)).';
        gbar = b(ii,:) .* conj(zz(ii,:).^jj(ii,:));
        MACi =      abs(sum(conj(gbar).*g   )) ...
             ./sqrt(abs(sum(conj(gbar).*gbar)) ...
                  .*abs(sum(conj(g   ).*g   )));
        MAC(irows,1) = MACi.';
    end;
end;

% compute (continuous-time) natural frequencies and damping
if (nargout>=6),
    m = 0; % will it always work with m=0, or do we have to play with it
    s = (log(poles) + 2*m*pi*sqrt(-1))/(k*T);
    wn = sqrt(real(s.*conj(s)));
    if (nargout>=7),
        zn = -real(s)./sqrt(wn);
    end;
end;
```

8.3.2 `era_test.m` — Simple ERA Example

`era_test` runs an example using `era` to identify a simple system.

```
% era_test.m

% this script runs a simple identification example using the
% Eigensystem Realization Algorithm via the era.m m-file.


% construct a simple 4 degree of freedom system, with
% 2 inputs and 3 outputs (not collocated).

n=4;   nrefs=2;   no=3;
T = .6;

[a,b,c,d,M,C,K,PP,1,W,Z] = ndof(n,T);
[a,b,c,d] = ssselect(a,b,c,d,size(b,2)+(1-nrefs:0),1:no);


% get the pulse response
nt = dtimvec(a,b,c,zeros(size(a,1),1),1e-3);
resp = zeros(nt,no*nrefs);
for k=1:nrefs,
    resp(:,(k-1)*no+(1:no)) = dimpulse(a,b,c,d,k,nt);
end;

% add a little noise
resp_noisy = resp + sqrt(mean(resp(:).^2))*randn(size(resp))/10;

% run ERA
[A,B,C,poles,mshapes,wn,zn,MAC,nout] = era(resp_noisy,T,nrefs, ...
                                          'ask',[16;25;40;80]*[1 1]);

% display natural frequencies and damping
disp('Exact natural frequencies and damping ratios');
[junk,ii] = sort(W);
disp([W(ii) Z(ii)]);

disp(' ');
disp('Estimated natural frequencies and damping ratios');
[junk,ii] = sort(wn);
disp([wn(ii) zn(ii)]);
```

8.4 TWO-STAGE ADAPTIVE MONITORING CODES

8.4.1 `rarx_test.m` — Evaluating a Two-Stage Adaptive Monitoring

`rarx_test` tests one example of a two-stage monitoring algorithm, using the "forgetting factor" variant of the recursive least-squares identification with an ARX model of the system.

```
function [h,p,yh] = rarx_test(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,  ...
                              a15,a16,a17,a18,a19,a20,a21,a22,a23,a24,a25,a26, ...
                              a27,a28,a29,a30,a31,a32,a33,a34,a35,a36,a37,a38,a39)
% RARX_TEST  Uses RARX to identify various NDOF systems.
%
%   RARX_TEST simulates a (possibly time-varying) n-degree-of-freedom system
%   with sensor noise and uses RARX (Recursive ARX) to identify natural
%   frequencies and damping ratios.
%
%   The arguments are in variable/value pairs.  For example, RARX_TEST('n',6)
%   sets the variable n (# of degrees of freedom) to 6.  Look at the code for
%   explanations of the variables and their default values.
%
%   See also RARX_TEST_RUN, RARX.

% Copyright (c)1996, Erik A. Johnson <johnsone@uiuc.edu>, 7/8/96

% variables and their default values
n = 1;                      % # of degrees-of-freedom
ni = 1;                     % # of inputs
no = 1;                     % # of outputs; no>1 won't work w/RARX -- MISO systems only
dt = 0.6;                   % time step
t0 = 0;                     % initial time
tf = 750;                   % length of simulation in seconds
dm = 0.0;                   % mass gradient
dc = 0.0;                   % damping gradient
dk = 0.0;                   % stiffness gradient
nu =  0;                    % number of sinusoids in input; use 0 for random
filt = 8;                   % order of output low-pass filter; use 0 for no filter
filtcut = 0.6;              % filter cutoff freq = filtcut/(2*dt)
noisemag = 5e-2;            % sensor noise (rms_noise/rms_signal)
outtype = 'displacement'; % sensor type
ff = .97;                   % forgetting factor; should be in [.97,.999]
modalresponse = [];         % should we output modal responses

% parse arguments
if rem(nargin,2),
    error('RARX_TEST requires ''variable''/value pairs.');
end;
for k=1:nargin/2,
    eval(['a' num2str(k*2-1)]) '= a' num2str(k*2) ';']);
end;

% number of time steps
nt = round(tf/dt);

% adjust filter parameter lengths
if isempty(filt), filt=8; end;
if isempty(filtcut), filtcut=.6; end;
if (length(filt)~=length(filtcut)),
    if (length(filt)==1),
        filt = filt*ones(size(filtcut));
    elseif (length(filtcut)==1),
        filtcut = filtcut*ones(size(filt));
    else,
        error('FILT and FILTCUT must be scalar or the same size.');
    end;
end;

% set up time vector
t = t0 + (0:nt-1).'*dt;

% set up inputs and outputs
```

rarx_test.m — Evaluating a Two-Stage Adaptive Monitoring (cont.)

```
inputs  = n-ni+1:n;
outputs = 1:no;

% set up force matrix u
if (nu>0),
    um = randn(nu,ni);
    uw = rand(nu,ni)*2;
    u = zeros(nt,ni);
    for kk=1:nu,
        u = u + um(kk*ones(nt,1),:).*sin(t*uw(kk,:));
    end;
    urms = sqrt(sum(u.^2)/nu);
    u = u ./ urms(ones(nt,1),:);
else,
    u = randn(nt,ni);
end;

% state initial conditions
x0 = zeros(2*n,1);

% do the integration through time
if all(dm(:)==0)&all(dc(:)==0)&all(dk(:)==0),
    % time-invariant problems
    [a,b,c,d] = ndof(n,dt,outtype);
    [a,b,c,d] = ssselect(a,b,c,d,inputs,outputs);
    [y,x] = dlsim(a,b,c,d,u,x0);
    xx = x(nt,:).';
    r = eig(a);
    r = r(:,ones(1,nt));
else,
    % set up mass, damping, and stiffness values
    if isstr(dm), dm=eval(dm); end;
    if (length(dm)==1),
        dm=dm*rem(floor(t/(tf/3)),2);
    elseif (length(dm)~=length(t)),
        error('DM must be scalar, a string, or the same length as the time vector.');
    end;
    if isstr(dc), dc=eval(dc); end;
    if (length(dc)==1),
        dc=dc*rem(floor(t/(tf/3)),2);
    elseif (length(dc)~=length(t)),
        error('DC must be scalar, a string, or the same length as the time vector.');
    end;
    if isstr(dk), dk=eval(dk); end;
    if (length(dk)==1),
        dk=dk*rem(floor(t/(tf/3)),2);
    elseif (length(dk)~=length(t)),
        error('DK must be scalar, a string, or the same length as the time vector.');
    end;
    mfract=ones(n,1); cfract=ones(n,1); kfract=ones(n,1); ifr=ceil(n/2);
    % check for piecewise time-invariants
    ii = [1; find(diff(dm(:))|diff(dk(:))|diff(dc(:)))+1; nt];
    if ~isempty(ii), if any(diff(ii)<10), ii=ones(1,1000); end; end;
    if (length(ii)<=10),
        y = zeros(nt,no);
        x = zeros(nt,2*n);
        r = zeros(2*n,nt);
        x(1,:) = x0(:).';
        for kkk=1:length(ii)-1,
            jj = ii(kkk):ii(kkk+1);
            jjj = (ii(kkk)+ii(kkk+1))/2;
            mfract(ifr)=1-dm(jjj); cfract(ifr)=1-dc(jjj); kfract(ifr)=1-dk(jjj);
            [a,b,c,d]=ndof(n,dt,outtype,mfract,cfract,kfract);
            [a,b,c,d] = ssselect(a,b,c,d,inputs,outputs);
            [y(jj,:),x(jj,:)] = dlsim(a,b,c,d,u(jj,:),x(ii(kkk),:));
            rr = eig(a);
            r(:,jj) = rr(:,ones(1,ii(kkk+1)-ii(kkk)+1));
        end;
    else,
        % initialize the data
```

rarx_test.m — Evaluating a Two-Stage Adaptive Monitoring (cont.)

```
        u = u.';
        xx = x0;
        y = zeros(no,nt);
        x = zeros(2*n,nt);
        r = zeros(2*n,nt);
        % do the integration
        [a,b,c,d] = ndof(n,dt,outtype);
        [a,b,c,d] = ssselect(a,b,c,d,inputs,outputs);
        for k=1:nt,
            y(:,k) = c*xx + d*u(:,k);
            x(:,k) = xx;
            mfract(ifr)=1-dm(k); cfract(ifr)=1-dc(k); kfract(ifr)=1-dk(k);
            [a,b,c,d] = ndof(n,dt,outtype,mfract,cfract,kfract);
            [a,b,c,d] = ssselect(a,b,c,d,inputs,outputs);
            r(:,k) = eig(a);
            xx = a*xx + b*u(:,k);
        end;
        % transpose the outputs
        u = u.';
        x = x.';
        y = y.';
    end;
end;

% convert to continuous time roots
r = log(r.')/dt;
% compute exact freqency and damping
[r,w_exact,z_exact] = thm2rts(r,'sort');

% set up the initial guess
[aa,bb,cc,dd] = ndof(n,dt,outtype);
[aa,bb,cc,dd] = ssselect(aa,bb,cc,dd,inputs,outputs);
[num,den] = ss2tf(aa,bb,cc,dd,1);
thm0 = [den(2:2*n+1) num(2:2*n+1) zeros(1,2*n*(ni-1))];
for nni=2:ni, num=ss2tf(aa,bb,cc,dd,nni); thm0((1:2*n)+nni*2*n)=num(2:2*n+1); end;

held=ishold;
co = get(gca,'ColorOrder');
coi = 0;
h = [];
p = [];
yh = [];
noise = randn(size(y)).*(ones(nt,1)*sqrt(sum(y.^2)/nt));
for nn=1:length(noisemag),
    % add some sensor noise
    yn = y + noise*noisemag(nn);
    un = u;

    yn_unfilt = yn;
    un_unfilt = un;
    for nfilt=1:length(filt),
        % optionally pass the output through a low-pass filter
        if (filt(nfilt)==0),
            yn = yn_unfilt;
            un = un_unfilt;
        else,
            [filtNum,filtDen] = butter(filt(nfilt),filtcut(nfilt));
            for kk=1:size(yn,2), yn(:,kk)=dlsim(filtNum,filtDen,yn_unfilt(:,kk)); end;
            for kk=1:size(un,2), un(:,kk)=dlsim(filtNum,filtDen,un_unfilt(:,kk)); end;
        end;

        % loop over various ff's
        for nf=1:length(ff),
            % do the identification
            norders = [2*n 2*n*ones(1,ni) ones(1,ni)];
            [thm,yhat]=rarx([yn un],norders,'ff',ff(nf),thm0);

            % do the output
            if ~isempty(modalresponse),
                % get the modal responses
```

rarx_test.m — Evaluating a Two-Stage Adaptive Monitoring (cont.)

```
                nvar=zeros(no); %assume nvar(0) ~= mean nvar
                nvar(:)=sum((yn(:,ones(no,1)*(1:no))-yhat(:,(1:no)'*ones(1,no))).^2)';
                [h1,p1,yh1] = rarx_kf([yn un],norders,thm,dt,modalresponse,nvar,yhat);
                h=[h h1];
                p=[p p1];
                yh=[yh yh1];
            else,
                % find the roots
                [r,w,z] = thm2rts(thm,2*n,dt,'sort');
                % plot the data
                h1=plot(t,w_exact,':',t,z_exact,':');
                hold('on');
                h2=plot(t,w,t,z);
                set([h1;h2],'Color',co(1+rem(coi,size(co,1)),:));
                coi = coi + 1;
                drawnow;
                h=[h;h1;h2];
            end;
        end;
    end;
end;
if (~held), hold('off'); end;
```

8.4.2 `rarx_test_run.m` — Evaluating Several Two-Stage Adaptive Monitoring Examples

`rarx_test_run` runs a number of tests of the two-stage adaptive monitoring using `rarx_test` and plots the results. The effects of various parameters (*e.g.*, the forgetting factor, noise magnitude, etc.) on one, two, and six degree of freedom systems are examined.

```
% rarx_test_run
%
% This does a bunch of runs of rarx_test with different arguments

scrn = 0;
c3 = [.4 .7 .99];
c4 = [.3 .53 .76 .99];
l3 = [1 1 .5];
l4 = [1 1 .5 .5];

if (scrn), ggg=[0 1]; else, ggg=[1 -1]; end;
ggg = [1 -1];% [0 1]; %[1 -1] for printouts


%%%%%%%%%%%%%%%
%             %
%  RARX stuff %
%             %
%%%%%%%%%%%%%%%

t = (0:750).';
plot(t,exp((750-t)*log([.95 .97 .98 .99 .995])));
axis([0 750 0 1]);
sxlabel('\times time [secs]'); sylabel('\times weight');

fn = '00_ff_on_rarx';
drawnow; if (scrn), pause; else, printsto('-deps',[fn '.eps']); end;
title(['Forgetting factor causes past data to be weighted exponentially smaller (' ...
       fn ')'],'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dps',[fn '.ps']); end;




%%%%%%%%%%%%%%%
%             %
%  SDOF SISO  %
%             %
%%%%%%%%%%%%%%%

% effect of ff on time-invariant problem
clf('reset');
h=rarx_test('n',1,'dm',0,'ff',[.95 .98 .99 .995]);
c=eval(['c' num2str(length(h)/4)]); l=eval(['l' num2str(length(h)/4)]);
for k=1:length(c), set(h((1:4)+(k-1)*4),'Color',[1 1 1]*(ggg(1)+ggg(2)*c(k)), ...
                                        'LineWidth',l(k)); end;
axis([0 750 .97 1.04])
[hax,hli,hte] = slegend('mouse',h(4:4:length(h)), ...
                        str2mat('\times\10{\italic\lambda} = 0.95', ...
                                '\times\10{\italic\lambda} = 0.98', ...
                                '\times\10{\italic\lambda} = 0.99', ...
                                '\times\10{\italic\lambda} = 0.995'));
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '01_ff_on_ti_siso_sdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of forgetting factor on time-invariant SDOF SISO (' fn ')'], ...
      'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

axis([0 750 .02 .1])
sylabel('\times damping ratio');
fn = '01_ff_on_ti_siso_sdof_z';
drawnow; pause;if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
```

rarx_test_run.m — Evaluating Several Two-Stage Adaptive Monitoring Examples (cont.)

```
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;



% effect of noise on time-invariant
clf('reset');
h=rarx_test('n',1,'dm',0,'ff',.98,'noisemag',[.25 .1 .05 .01]);
c=eval(['c' num2str(length(h)/4)]); l=eval(['l' num2str(length(h)/4)]);
for k=1:length(c), set(h((1:4)+(k-1)*4),'Color',[1 1 1]*(ggg(1)+ggg(2)*c(k)), ...
                                        'LineWidth',l(k)); end;
axis([0 750 .97 1.14]);
[hax,hli,hte] = slegend('mouse',h(4:4:length(h)), ...
                        str2mat('\times\10rms_{noise} = 0.25', ...
                        '\times\10rms_{noise} = 0.10', ...
                        '\times\10rms_{noise} = 0.05', ...
                        '\times\10rms_{noise} = 0.01'));
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '02_noise_on_ti_siso_sdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of sensor noise rms on time-invariant SDOF SISO (' fn ')'],'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

axis([0 750 .03 .2]);
sylabel('\times damping ratio');
fn = '02_noise_on_ti_siso_sdof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;



% effect of filtering on time-invariant
clf('reset');
h=rarx_test('n',1,'dm',0,'ff',.99,'filt',[0 1 4 8]);
c=eval(['c' num2str(length(h)/4)]); l=eval(['l' num2str(length(h)/4)]);
for k=1:length(c), set(h((1:4)+(k-1)*4),'Color',[1 1 1]*(ggg(1)+ggg(2)*c(k)), ...
                                        'LineWidth',l(k)); end;
axis([0 750 .97 1.04])
[hax,hli,hte] = slegend('mouse',h(4:4:length(h)), ...
                        str2mat('\times\10no filter', ...
                        '\times\10filter order = 1', ...
                        '\times\10filter order = 4', ...
                        '\times\10filter order = 8'));
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '03_filt_on_ti_siso_sdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of filtering on time-invariant SDOF SISO (' fn ')'],'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

axis([0 750 .03 .1])
sylabel('\times damping ratio');
fn = '03_filt_on_ti_siso_sdof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;



% effect of ff on piece-wise time-invariant (decrease mass)
clf('reset');
h=rarx_test('n',1,'dm','rem(floor(t/(tf/3)),2)/2','ff',[.95 .98 .99]);
```

rarx_test_run.m — Evaluating Several Two-Stage Adaptive Monitoring Examples (cont.)

```
c=eval(['c' num2str(length(h)/4)]); l=eval(['l' num2str(length(h)/4)]);
for k=1:length(c), set(h((1:4)+(k-1)*4),'Color',[1 1 1]*(ggg(1)+ggg(2)*c(k)), ...
                                        'LineWidth',l(k)); end;
axis([0 750 .9 1.5])
[hax,hli,hte] = slegend('mouse',h(4:4:length(h)), ...
                        str2mat('\times\10{\italic\lambda} = 0.95', ...
                                '\times\10{\italic\lambda} = 0.98', ...
                                '\times\10{\italic\lambda} = 0.99'));
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '04_ff_on_ptimm_siso_sdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of forgetting factor on piece-wise time-invariant SDOF SISO (' fn ')'], ...
      'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

axis([0 750 .02 .12])
sylabel('\times damping ratio');
fn = '04_ff_on_ptimm_siso_sdof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
```

```
% effect of ff on piece-wise time-invariant (increase mass)
clf('reset');
h=rarx_test('n',1,'dm','-rem(floor(t/(tf/3)),2)/2','ff',[.95 .98 .99]);
c=eval(['c' num2str(length(h)/4)]); l=eval(['l' num2str(length(h)/4)]);
for k=1:length(c), set(h((1:4)+(k-1)*4),'Color',[1 1 1]*(ggg(1)+ggg(2)*c(k)), ...
                                        'LineWidth',l(k)); end;
axis([0 750 .75 1.1])
[hax,hli,hte] = slegend('mouse',h(4:4:length(h)), ...
                        str2mat('\times\10{\italic\lambda} = 0.95', ...
                                '\times\10{\italic\lambda} = 0.98', ...
                                '\times\10{\italic\lambda} = 0.99'));
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '05_ff_on_ptimp_siso_sdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of forgetting factor on piece-wise time-invariant SDOF SISO (' fn ')'], ...
      'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
```

```
% effect of ff on piece-wise time-invariant (decrease frequency)
clf('reset');
h=rarx_test('n',1,'dm',0,'dk','rem(floor(t/(tf/3)),2)/2','dc', ...
            'rem(floor(t/(tf/3)),2)*(1-sqrt(.5))','ff',[.95 .98 .99]);
c=eval(['c' num2str(length(h)/4)]); l=eval(['l' num2str(length(h)/4)]);
for k=1:length(c), set(h((1:4)+(k-1)*4),'Color',[1 1 1]*(ggg(1)+ggg(2)*c(k)), ...
                                        'LineWidth',l(k)); end;
axis([0 750 .6 1.1])
[hax,hli,hte] = slegend('mouse',h(4:4:length(h)), ...
                        str2mat('\times\10{\italic\lambda} = 0.95', ...
                                '\times\10{\italic\lambda} = 0.98', ...
                                '\times\10{\italic\lambda} = 0.99'));
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '06_ff_on_ptiwm_siso_sdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of forgetting factor on piece-wise time-invariant SDOF SISO (' fn ')'], ...
      'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
```

rarx_test_run.m — Evaluating Several Two-Stage Adaptive Monitoring Examples (cont.)

```
axis([0 750 .01 .11])
sylabel('\times damping ratio');
fn = '06_ff_on_ptiwm_siso_sdof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;




% effect of ff on piece-wise time-invariant (increase frequency)
clf('reset');
h=rarx_test('n',1,'dm',0,'dk','-rem(floor(t/(tf/3)),2)','dc', ...
            '-rem(floor(t/(tf/3)),2)*(sqrt(2)-1)','ff',[.95 .98 .99]);
c=eval(['c' num2str(length(h)/4)]); l=eval(['l' num2str(length(h)/4)]);
for k=1:length(c), set(h((1:4)+(k-1)*4),'Color',[1 1 1]*(ggg(1)+ggg(2)*c(k)), ...
                                        'LineWidth',l(k)); end;
axis([0 750 .9 1.5])
[hax,hli,hte] = slegend('mouse',h(4:4:length(h)), ...
                        str2mat('\times\10{\italic\lambda} = 0.95', ...
                                '\times\10{\italic\lambda} = 0.98', ...
                                '\times\10{\italic\lambda} = 0.99'));
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '07_ff_on_ptiwp_siso_sdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of forgetting factor on piece-wise time-invariant SDOF SISO (' fn ')'], ...
      'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

axis([0 750 .02 .11])
sylabel('\times damping ratio');
fn = '07_ff_on_ptiwp_siso_sdof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;




% effect of ff on piece-wise time-invariant (decrease damping)
clf('reset');
h=rarx_test('n',1,'dm',0,'dc','rem(floor(t/(tf/3)),2)/2','ff',[.95 .98 .99]);
c=eval(['c' num2str(length(h)/4)]); l=eval(['l' num2str(length(h)/4)]);
for k=1:length(c), set(h((1:4)+(k-1)*4),'Color',[1 1 1]*(ggg(1)+ggg(2)*c(k)), ...
                                        'LineWidth',l(k)); end;
axis([0 750 .92 1.08])
[hax,hli,hte] = slegend('mouse',h(4:4:length(h)), ...
                        str2mat('\times\10{\italic\lambda} = 0.95', ...
                                '\times\10{\italic\lambda} = 0.98', ...
                                '\times\10{\italic\lambda} = 0.99'));
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '08_ff_on_ptizm_siso_sdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of forgetting factor on piece-wise time-invariant SDOF SISO (' fn ')'], ...
      'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

axis([0 750 .00 .1])
sylabel('\times damping ratio');
fn = '08_ff_on_ptizm_siso_sdof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
```

rarx_test_run.m — Evaluating Several Two-Stage Adaptive Monitoring Examples (cont.)

```
% effect of ff on piece-wise time-invariant (increase damping)
clf('reset');
h=rarx_test('n',1,'dm',0,'dc','-rem(floor(t/(tf/3)),2)/2','ff',[.95 .98 .99]);
c=eval(['c' num2str(length(h)/4)]); l=eval(['l' num2str(length(h)/4)]);
for k=1:length(c), set(h((1:4)+(k-1)*4),'Color',[1 1 1]*(ggg(1)+ggg(2)*c(k)), ...
                                        'LineWidth',l(k)); end;
axis([0 750 .92 1.08])
[hax,hli,hte] = slegend('mouse',h(4:4:length(h)), ...
                        str2mat('\times\10{\italic\lambda} = 0.95', ...
                                '\times\10{\italic\lambda} = 0.98', ...
                                '\times\10{\italic\lambda} = 0.99'));
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '09_ff_on_ptizp_siso_sdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of forgetting factor on piece-wise time-invariant SDOF SISO (' fn ')'], ...
      'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

axis([0 750 .02 .13])
sylabel('\times damping ratio');
fn = '09_ff_on_ptizp_siso_sdof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
```

```
% effect of ff on continuously time-varying (mass varies)
clf('reset');
h=rarx_test('n',1,'ff',[.95 .98 .99],'tf',2250, ...
            'dm','1-1./(1+(t>=(tf/3)).*sin((t/(tf/3)-1)*pi)*(sqrt(2)-1)).^2');
c=eval(['c' num2str(length(h)/4)]); l=eval(['l' num2str(length(h)/4)]);
for k=1:length(c), set(h((1:4)+(k-1)*4),'Color',[1 1 1]*(ggg(1)+ggg(2)*c(k)), ...
                                        'LineWidth',l(k)); end;
axis([0 2250 .5 1.5])
[hax,hli,hte] = slegend('mouse',h(4:4:length(h)), ...
                        str2mat('\times\10{\italic\lambda} = 0.95', ...
                                '\times\10{\italic\lambda} = 0.98', ...
                                '\times\10{\italic\lambda} = 0.99'));
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '10_ff_on_tvm_siso_sdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of forgetting factor on continuously time-varying SDOF SISO (' fn ')'], ...
      'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

axis([0 2250 .02 .12])
sylabel('\times damping ratio');
fn = '10_ff_on_tvm_siso_sdof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
```

```
% effect of ff on continuously time-varying (mass varies), accel output
```

rarx_test_run.m — Evaluating Several Two-Stage Adaptive Monitoring Examples (cont.)

```
clf('reset');
h=rarx_test('n',1,'ff',[.95 .98 .99],'tf',2250,'outtype','acceleration', ...
            'dm','1-1./(1+(t>=(tf/3)).*sin((t/(tf/3)-1)*pi)*(sqrt(2)-1)).^2');
c=eval(['c' num2str(length(h)/4)]); l=eval(['l' num2str(length(h)/4)]);
for k=1:length(c), set(h((1:4)+(k-1)*4),'Color',[1 1 1]*(ggg(1)+ggg(2)*c(k)), ...
                                        'LineWidth',l(k)); end;
axis([0 2250 .9 1.5])
[hax,hli,hte] = slegend('mouse',h(4:4:length(h)), ...
                        str2mat('\times\10{\italic\lambda} = 0.95', ...
                                '\times\10{\italic\lambda} = 0.98', ...
                                '\times\10{\italic\lambda} = 0.99'));
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '11_ff_on_tvma_siso_sdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of ff on continuously time-varying SDOF SISO (' fn ')'],'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

axis([0 2250 .02 .12])
sylabel('\times damping ratio');
fn = '11_ff_on_tvma_siso_sdof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
```

```
% effect of ff on continuously time-varying (damping varies)
clf('reset');
h=rarx_test('n',1,'ff',[.95 .98 .99],'tf',2250, ...
            'dc','-(t>=(tf/3)).*sin((t/(tf/3)-1)*pi)/2');
c=eval(['c' num2str(length(h)/4)]); l=eval(['l' num2str(length(h)/4)]);
for k=1:length(c), set(h((1:4)+(k-1)*4),'Color',[1 1 1]*(ggg(1)+ggg(2)*c(k)), ...
                                        'LineWidth',l(k)); end;
axis([0 2250 .95 1.05])
[hax,hli,hte] = slegend('mouse',h(4:4:length(h)), ...
                        str2mat('\times\10{\italic\lambda} = 0.95', ...
                                '\times\10{\italic\lambda} = 0.98', ...
                                '\times\10{\italic\lambda} = 0.99'));
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '12_ff_on_tvz_siso_sdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of forgetting factor on continuously time-varying SDOF SISO (' fn ')'], ...
      'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

axis([0 2250 .00 .12])
sylabel('\times damping ratio');
fn = '12_ff_on_tvz_siso_sdof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
```

```
%%%%%%%%%%%%%%%
%             %
%  2DOF MISO  %
%             %
%%%%%%%%%%%%%%%
```

rarx_test_run.m — Evaluating Several Two-Stage Adaptive Monitoring Examples (cont.)

```
% piecewise time-invariant problem (mass increases)
clf('reset');
h=rarx_test('n',2,'noisemag',5e-3,'dm','rem(floor(t/(tf/3)),2)/2','ff',[.95 .98 .99]);
set(h,'Color',[1 1 1]);
axis([0 750 .4 1.8])
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '22_tv_miso_2dof_w';
set(h([3:4 7:8]),'Visible','off');
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Piece-wise time-invariant 2DOF SISO (' fn ')'],'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

axis([0 750 0 .12])
sylabel('\times damping ratio');
set(h,'Visible','on');
set(h([1:2 5:6]),'Visible','off');
fn = '22_tv_miso_2dof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;

axis([0 750 .02 .1])
fn = '22_tv_miso_2dof_z2';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Piece-wise time-invariant 2DOF SISO (' fn ')'],'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
```

```
%%%%%%%%%%%%%%%%
%              %
%  6DOF MISO   %
%              %
%%%%%%%%%%%%%%%%
```

```
% time-invariant problem
clf('reset');
h=rarx_test('n',6,'noisemag',3e-6,'ni',6,'dm',0,'ff',.99);
co=get(gca,'ColorOrder'); l=.5*(1+rem(1:length(co),2));
for k=1:length(h), set(h(k),'Color',co(1+rem(k-1,size(co,1)),:),'LineWidth',l(k)); end;
axis([0 750 0 2.2])
set(h([7:12 19:24]),'Visible','off');
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '13_ti_miso_mdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of forgetting factor on time-invariant MDOF MISO (' fn ')'], ...
      'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

set(h,'Visible','on');
set(h([1:6 13:18]),'Visible','off');
axis([0 750 0 .11]);
sylabel('\times damping ratio');
fn = '13_ti_miso_mdof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;


% piecewise time-invariant problem
clf('reset');
```

rarx_test_run.m — Evaluating Several Two-Stage Adaptive Monitoring Examples (cont.)

```
h=rarx_test('n',6,'tf',2250,'noisemag',3e-6,'ni',6, ...
            'dm','rem(floor(t/(tf/3)),2)/2','ff',.99);
co=get(gca,'ColorOrder'); l=.5*(1+rem(1:length(co),2));
for k=1:length(h), set(h(k),'Color',co(1+rem(k-1,size(co,1)),:),'LineWidth',l(k)); end;
axis([0 2125 0 3])
set(h([7:12 19:24]),'Visible','off');
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '22_pti_miso_mdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of forgetting factor on time-invariant MDOF MISO (' fn ')'], ...
      'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

set(h,'Visible','on');
set(h([1:6 13:18]),'Visible','off');
axis([0 2125 0 .13]);
sylabel('\times damping ratio');
fn = '22_pti_miso_mdof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;

% check modal responses
clf('reset');
dt=.6; tf=2250; t=(0:round(tf/dt)-1).'*dt; n=6;
[x,p] = rarx_test('n',n,'tf',2250,'dt',.6,'noisemag',3e-6,'ni',n, ...
                  'modalresponse','ceil(tf/dt/10)', ...
                  'dm','rem(floor(t/(tf/3)),2)/2','ff',.99);
nt = size(x,1);
h=plot((0:nt-1)*dt,x(:,1:2:2*n));
co=get(gca,'ColorOrder'); l=.5*(1+rem(1:length(co),2));
for k=1:length(h), set(h(k),'Color',co(1+rem(k-1,size(co,1)),:),'LineWidth',l(k)); end;
set(gca,'XLim',[0 tf]);
sxlabel('\times time [secs]'); sylabel('\times modal response');

fn = ['23_' num2str(n) 'dof_modalresp'];
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Modal response of piecewise time-invariant ' num2str(n) ...
       'DOF MISO (' fn ')'],'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

% rms of modal responses
clf('reset');
xrms = sqrt(sum(x(:,1:2:2*n).^2)/nt)';
h=semilogy(1:n,xrms,'w-',1:n,xrms,'w-');
set(gca,'XTick',(1:n)','XLim',[1 n]);
sxlabel('\times mode number'); sylabel('\times modal response RMS');

fn = ['24_' num2str(n) 'dof_modalresp_rms'];
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['RMS of modal responses of piecewise time-invariant ' num2str(n) ...
       'DOF MISO (' fn ')'],'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

% continuously time-varying problem (mass)

% Note:  On limited-memory platforms (e.g., PC or Mac), the following actually
%        must be done in segments since the call to RARX takes too much memory.
%        See RARX_PIECEWISE for details on how that may be done.

clf('reset');
h=rarx_test('n',6,'tf',2250,'noisemag',1e-6,'ni',6,'ff',.99, ...
```

rarx_test_run.m — Evaluating Several Two-Stage Adaptive Monitoring Examples (cont.)

```
            'dm','1-1./(1+(t>=(tf/3)).*sin((t/(tf/3)-1)*pi)*(sqrt(2)-1)).^2');
co=get(gca,'ColorOrder');
for k=1:length(h), set(h(k),'Color',co(1+rem(k-1,size(co,1)),:)); end;
axis([0 750 0 2.2])
set(h([7:12 19:24]),'Visible','off');
sxlabel('\times time [secs]'); sylabel('\times frequency [\frac{rads}{sec}]');

fn = '14_ti_miso_mdof_w';
drawnow; pause; if (~scrn), printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
title(['Effect of forgetting factor on time-varying MDOF MISO (' fn ')'], ...
      'FontName','Times');
drawnow; if (scrn), pause; else, printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;

set(h,'Visible','on');
set(h([1:6 13:18]),'Visible','off');
axis([0 750 0 .11]);
sylabel('\times damping ratio');
fn = '14_ti_miso_mdof_z';
drawnow; pause; if (~scrn), printsto('-dpsc',[fn '.ps']); stfixps([fn '.ps']); end;
title('');
drawnow; if (scrn), pause; else, printsto('-depsc',[fn '.eps']); stfixps([fn '.eps']); end;
```

8.4.3 `thm2rts.m` — Convert `rarx` Identified Model to Modal Characteristics

`thm2rts` converts the identification model output by `rarx` to natural frequencies and damping ratios.

```
function [r,w,z] = thm2rts(thm,n,dt,dosort)
% THM2RTS compute roots (& freq./damping) from recursive SysID THM matrix.
%
%   [R,W,Z] = THM2RTS(THM,N,T) returns the continuous-time roots R,
%             and modal frequencies (W) and damping ratios (Z), for
%             the THM matrix returned by the recursive system
%             identification functions, where N is the order of the
%             system (= # of resulting roots = # of cols of THM used).
%
%       If THM is M-by-(N+K), then R is M-by-N, and W and Z are
%       M-by-ceil(N/2), where pairs of overdamped roots show up as
%       NaN's in W and Z.
%
%       T is the sampling period.
%
%  [R,W,Z] = THM2RTS(R) uses the given continuous-time roots to compute
%                       modal frequencies and damping ratios.
%
%  [R,W,Z] = THM2RTS(...,'sort') will sort R, W, and Z by frequency at
%                                 each time step.
%
%  See also RARX, RARMAX, etc.

% Copyright (c)1996, Erik A. Johnson <johnsone@uiuc.edu>, 6/3/96
% added sorting, 6/8/96

% check args
if (nargin<1), error('THM2RTS requires at least 1 input argument.'); end;
if (nargin>4), error('THM2RTS takes at most 4 input arguments.'); end;

% do the work
if (nargin <= 2),
    % we have the roots
    r = thm;
    if (nargin==2), dosort=n; else, dosort=[]; end;
    [m,n] = size(r);
    rr = r.';
else,
    % find the roots
    m = size(thm,1);
    r = zeros(n,m);
    for k=1:m,
        r(:,k) = roots([1 thm(k,1:n)]);
    end;

    % convert to continuous time
    r = log(r)/dt;
    rr = r;
    r = r.';
    if (nargin==3), dosort=[]; end;
end;

% sort roots into complex pairs and real values
if (any(imag(rr(:))==0)),
    [junk,ii] = sort(-abs(imag(rr)));
    ii = ii + n*ones(n,1)*(0:m-1);
    rr(:) = rr(ii(:));
    rr(imag(rr)==0) = rr(imag(rr)==0) * NaN;
end;
rr = rr.';

% compute fequency and damping
if (rem(n,2)), rr=[rr NaN*ones(m,1)]; n=n+1; end;
w = real(sqrt(rr(:,1:2:n).*rr(:,2:2:n)));
z = -real(rr(:,1:2:n)+rr(:,2:2:n))/2./w;
```

thm2rts.m — Convert rarx Identified Model to Modal Characteristics (cont.)

```
% sort them
if isempty(dosort), dosort=0; end;
if ((isstr(dosort)|any(dosort)) & (n>2)),
    [w,ii] = sort(w.');
    z = z.';
    ii = ii + n/2*ones(n/2,1)*(0:m-1);
    z(:) = z(ii);
    w = w.';
    z = z.';
end;
```

8.4.4 `ss2modal.m` — Convert General State-Space System to Modal State-Space

This function converts a general state-space system to a modal state-space system whose states are modal displacement and velocities for continuous- or discrete-time systems.

```
function [a,b,c,tm,p]=ss2modal(a,b,c,t,dosort)
% SS2MODAL  Convert general state-space system to modal state-space.
%
%   [Am,Bm,Cm] = SS2MODAL(A,B,C) converts the given general, continuous-time,
%                                state-space system to modal state coordinates.
%
%   [Am,Bm,Cm,Tm] = SS2MODAL(A,B,C) also returns the similarity transformation
%                                    matrix Tm used in the conversion
%                                    (if the old states are X and the new
%                                    modal states are Q, then X=Tm*Q,
%                                    Am=inv(Tm)*A*Tm, Bm=inv(Tm)*B, C=C*Tm).
%
%   [Am,Bm,Cm,Tm,P] = SS2MODAL(A,B,C) also returns the number of underdamped
%                                      eigenvalue pairs in P.
%
%   SS2MODAL(A,B,C,T) performs the same operations for a discrete-time system
%                     with sample time T.  An empty T implies continuous-time.
%
%   SS2MODAL(A,B,C,T,'sort') rearranges the states such that the underdamped
%                            modes (those with complex eigenvalue pairs) are
%                            grouped first by increasing frequency, then the
%                            over-damped modes.
%
%   Note that the output of SS2MODAL uses the modal coordinates and their
%   derivatives as the states, whereas the CANON(...,'modal') uses some
%   transformation of the states of SS2MODAL.
%
%  See also CDF2RDF, CANON.

% Copyright (c)1996, Erik A. Johnson <johnsone@uiuc.edu>, 4/10/96

% handle the inputs and outputs
if (nargin<1), error('SS2MODAL requires at least 1 argument, the A matrix.');
elseif (nargin>5), error('SS2MODAL takes at most 5 arguments.');
elseif (nargout>5), error('SS2MODAL returns at most 5 outputs.');
elseif ((nargin<3)&(nargout>nargin)),
    error('SS2MODAL cannot convert matrices not supplied.');
elseif (nargin==1), error(abcdchk(a));
elseif (nargin==2), error(abcdchk(a,b));
else, error(abcdchk(a,b,c));
end;

% handle default values
if (nargin<4), t=[]; end;
if(nargin<5),dosort=[];end; if(isempty(dosort)),dosort=0;end;

% solve eigensystem
[Phi,D]=eig(a);
D=diag(D);
n=length(D);

% handle complex eigenvalues (eig() leaves them in pairs next to each other)
complexlist=(imag(D)~=0);
ii=find(rem(cumsum(complexlist).*(complexlist),2)==1);
p = length(ii);

% convert discrete-time eigenvalues to corresponding continuous time
if (~isempty(t)),
    Dmask = (D==0);
    D(Dmask) = D(Dmask) - inf;
    D(~Dmask) = log(D(~Dmask))/t;
end;

% compute the similarity transformation matrix
if (p>0),
    z00=ones(n,1);    z00(ii)=-D(ii+1);
```

ss2modal.m — Convert General State-Space System to Modal State-Space (cont.)

```
    zm1=zeros(n-1,1); zm1(ii)=-D(ii);
    zp1=zeros(n-1,1); zp1(ii)=ones(length(ii),1);
    Z = diag(z00) + diag(zm1,-1) + diag(zp1,1);
else, % no transformation
    Z = eye(n);
end;

% combine with eigenmatrix
tm=real(Phi*Z);
rcond(tm);
invtm=inv(tm);

% sort by increasing frequency
if (dosort & p>0),
    [junk,jj]=sort(real(D(ii).*D(ii+1)));
    jj=[ii(jj) ii(jj)+1].'; jj=[jj(:);find(~complexlist)];
    tm = tm(:,jj);
    invtm = invtm(jj,:);
end;

% compute them
a=invtm*a*tm;
if (nargout>1),
    b=invtm*b;
    if (nargout>2),
        c=c*tm;
    end;
end;
```

8.4.5 `rarx_kf.m` — On-line Monitoring via a Kalman Filter

`rarx_kf` uses the results of an `rarx` identification and the input/output data to compute modal displacements and velocities. An entire time history is computed here at once, but that is merely for convenience; in a real-world system, this would run on-line in parallel with the identification routine.

```
function [xhat,p,yhat] = rarx_kf(z,nn,thm,T,howoften,nvar,yhat,xhat0,phat0)
% RARX_KF  Compute modal responses from RARX model using Kalman filter.
%
%    [XM,P,YHAT2] = RARX_KF([Y U],NN,THM,T,HOWOFTEN,NVAR,YHAT,XHAT0,PHAT0)
%
%             uses a Kalman filter toestimate modal responses of the system
%             with inputs U, output Y, and RARX-estimated system matrices THM.
%             Note that like RARX, this only handles single-output systems;
%             thus Y is a column vector.
%
%             NN is the same matrix passed to RARX that specifies the orders
%             and delay [na nb nk] of the ARX model.  (Since this is single-
%             output, NN must be a row vector.)
%
%             T is the sampling time.
%
%             HOWOFTEN (optional) is a scalar integer that specifies the
%             number of time steps between updating the (A,B,C,D) model of the
%             system, used by the Kalman filter, from the THM argument.  If
%             empty or not supplied, its value is one (i.e., update every time
%             step).
%
%             NVAR (optional) is the noise variance.  It is assumed to be unity
%             if empty or not supplied.  A constant variance may be specified
%             with a scalar NVAR.
%
%             YHAT (optional) is an estimate of the system output, generally
%             that returned by RARX.  If YHAT is empty or not supplied, then Y
%             is used instead.
%
%             XHAT0 (optional) and PHAT0 (optional) are the initial state and
%             state covariances, respectively.  If empty or not supplied, they
%             default to zero.
%
%             The inputs Y, U, THM, (if supplied) NVAR, and (if supplied) YHAT
%             should all have the same number of rows, and YHAT (if supplied)
%             should be the same size as Y.
%
%             The output XM will have the same number of rows as Y and one
%             column per state.  The first 2*P columns are true modes (complex
%             eigenvalues) (displacement first, then velocity), sorted by
%             increasing frequency.
%
%             The output YHAT2 is the same size as Y and is the estimate of
%             the system output with no noise.
%
%   See also RARX, ARX, RARX_TEST.

% Copyright (c)1996, Erik A. Johnson <johnsone@uiuc.edu>, 7/8/96

% check # of arguments
if (nargin<3), error('RARX_KF requires at least 3 input arguments.');
elseif (nargin>9), error('RARX_KF takes at most 9 input arguments.');
elseif (nargout>4), error('RARX_KF produces at most 4 outputs.');
end;

% check argument sizes
no = 1;
[nt,ni] = size(z);
ni = ni-no;
[n1,nthm] = size(thm);
if (nargin<4), T=[]; end;
if (isempty(T)), T=1; else, T=abs(T(:)); end;
```

rarx_kf.m — On-line Monitoring via a Kalman Filter (cont.)

```
if (nargin<5), howoften=[]; end;
if (isempty(howoften)), howoften=1; else, howoften=max(1,round(howoften(1))); end;
if (nargin<6), nvar=[]; end;
if (isempty(nvar)), nvar=1; end;
if (all(size(nvar)>1)),
    error('RARX_KF requires that NVAR be a column vector or scalar.');
end;
if (length(nvar)==1), nvar=nvar*ones(nt,1); else, nvar=nvar(:); end;
if (nargin<7), yhat=[]; end;
if (isempty(yhat)),
    n2 = nt;
else,
    [n2,nyhat] = size(yhat);
    if (no~=nyhat),
        error('RARX_KF requires that YHAT be empty or the same size as Y.');
    end;
end;
if any(nt~=[n1 n2 length(nvar)]),
    error('RARX_KF requires that [Y U], THM, and YHAT have the same number of rows.');
end;
if (nargin<8), xhat0=[]; end;
if (nargin<9), phat0=[]; end;

% check order sizes
if (size(nn,1)~=no),
    error('RARX_KF requires that NN have the same # of rows as Y has columns.');
end;
if (size(nn,2)-no~=2*ni),
    error('RARX_KF requires that NN have 2*(#cols of u)+(#cols of y) columns.');
end;

% preparation to get models
na=nn(1); nb=nn(2:1+ni); nk=nn(2+ni:1+2*ni);
if any(nk<0),
    error('RARX_KF cannot deal with non-causal systems (i.e., nk<0).');
end;
n = max(na,max(nb+nk)-1);
haved = nb>0 & nk==0;
nbi = nb - haved;
bk = (1:max(nbi))';
bk = bk(:,ones(1,ni));
bj = n*(0:ni-1) + nk + (nk==0) - 1;
bj = bk + bj(ones(max(nbi),1),:);
nbii = nbi(ones(max(nbi),1),:);
bi = bj(bk(:)<=nbii(:));

Dj = [zeros(1,min(ni,1)) cumsum(nb(1:length(nb)-1))] + na;
Bj = Dj + haved;
Bj = bk + Bj(ones(max(nbi),1),:);

Bi = Bj(bk(:)<=nbii(:));

di = find(haved);
Di = Dj(haved) + 1;


% get the first system model in observer canonical form
k = 1;
a = diag(ones(n-1,1),1);
a(1:na,1) = -thm(k,1:na).';
b = zeros(n,ni);
if all(size(bi)), b(bi)=thm(k,Bi); end;
c = [1 zeros(1,n-1)];
d = zeros(1,ni);
if all(size(di)),
    d(di)=thm(k,Di);
    b = b + a(:,1)*d;
end;
%%%%%[a,b,c,d] = minreal(a,b,c,d);
[a,b,c,tm,p1]=ss2modal(a,b,c,T,'dosort');
```

rarx_kf.m — On-line Monitoring via a Kalman Filter (cont.)

```
% create some space
if (~isempty(yhat)), y=yhat(:); else, y=z(:,1:no); end;
u = z(:,no+(1:ni));
yhat = zeros(nt,no);
xhat = zeros(nt,n);
p    = zeros(nt,1);
p(1) = p1;

% handle initial conditions
if (~isempty(xhat0)),
    if (prod(size(xhat0))~=n),
        error('RARX_KF requires that XHAT0 be a column vector of length NN(1).');
    end;
    xhat(1,:) = xhat0(:).';
end;
if (~isempty(phat0)),
    if (any(size(phat0)~=n)),
        error('RARX_KF requires that XHAT0 be a NN(1)-by-NN(1) matrix.');
    end;
    phat = phat0;
else,
    phat = zeros(n);
end;

% loop over the times
for k=2:nt,
    xbar = a*xhat(k-1,:).' + b*u(k-1,:).';
    pbar = a*phat*a.';
    if (rem(k-1,howoften)==0),
        % update the KF model
        a = diag(ones(n-1,1),1);
        a(1:na,1) = -thm(k,1:na).';
        b = zeros(n,ni);
        if all(size(bi)), b(bi)=thm(k,Bi); end;
        c = [1 zeros(1,n-1)];
        d = zeros(1,ni);
        if all(size(di)),
            d(di)=thm(k,Di);
            b = b + a(:,1)*d;
        end;
        %%%%%[a,b,c,d] = minreal(a,b,c,d);
        [a,b,c,tm,p1]=ss2modal(a,b,c,T,'dosort');
    end;
    g = pbar*c.'/(c*pbar*c.'+nvar(k));
    phat = (eye(n)-g*c)*pbar;
    xhat(k,:) = ((eye(n)-g*c)*xbar-g*d*u(k,:).'+g*y(k,:).').';
    yhat(k,:) = (c*xhat(k,:).'+d*u(k,:).');
    p(k) = p1;
end;
```

8.4.6 `rarx_piecewise.m` — RARX Identification in Segments

`rarx_piecewise` calls `rarx` to do identification of a segment its data. This is useful for doing simulation of systems over a longer time period on limited memory platforms. Where memory use is not an issue, or the time sequence is short, use `rarx` instead.

```
function [thm,yhat,P,Phi] = rarx_piecewise(tmax,z,nn,adm,adg,thm0,P0,Phi0)
% RARX_PIECEWISE  Do RARX identification in segments.
%
%    RARX_PIECEWISE(TMAX,Z,NN,ADM,ADG,...) does the same thing as RARX
%                                          except in segments over time.
%               This is convenient for platforms (e.g. PC or Mac) where
%               memory is limited.  TMAX is the amount of time (in seconds)
%               between updates of the "waitbar" put on the screen while
%               it is running.
%
%               The outputs and the remaining inputs are identical to
%               those of RARX.
%
%   See also RARX.

% Copyright (c)1996, Erik A. Johnson <johnsone@uiuc.edu>, 7/8/96

% check # of arguments.
if (nargin<5), error('RARX_PIECEWISE requires at least 5 input arguments.'); end;
if isempty(tmax), tmax=10; end;

% check sizes and handle degenerate case
nt = size(z,1);
if (nt==0), thm=[]; yhat=[]; P=[]; Phi=[]; return; end;

% do the first segments
h = waitbar(0,'Please wait...');
t0=cputime;
    if (nargin==5), [thm1,yhat1,P,Phi] = rarx(z(1,:),nn,adm,adg);
elseif (nargin==6), [thm1,yhat1,P,Phi] = rarx(z(1,:),nn,adm,adg,thm0);
elseif (nargin==7), [thm1,yhat1,P,Phi] = rarx(z(1,:),nn,adm,adg,thm0,P0);
elseif (nargin>=8), [thm1,yhat1,P,Phi] = rarx(z(1,:),nn,adm,adg,thm0,P0,Phi0);
end;
t1=cputime; waitbar(1/nt);

% allocate some memory for the remaining segments
thm  =zeros(nt,length(thm1));   thm(1,:)=thm1;
yhat =zeros(nt,length(yhat1)); yhat(1,:)=yhat1;

% loop over remaining segments
num = 1;
mag = 2;
did = 1;
while (did < nt),
    % adaptively adjust the length of the segments to approximate tmax
    if (mag>1),
        if (t1-t0<tmax),
            num = num * mag;
        else,
            num = max(1,floor(num*tmax/(t1-t0+eps)));
            mag = 1;
        end;
    end;
    % do the next segment
    num = min(nt-did,num);
    t0=cputime;
    [thm1,yhat1,P,Phi] = rarx(z(did+(1:num),:),nn,adm,adg,thm1(size(thm1,1),:),P,Phi);
    t1=cputime;
    % store the results in our outputs
    thm(did+(1:num),:)=thm1; yhat(did+(1:num),:)=yhat1;
    did = did + num;
    waitbar(did/nt);
end;
waitbar(1); close(h);
```

9.0 APPENDIX B: ABSTRACTS RELATED TO H_∞-BASED IDENTIFICATION

A table of many of the papers related to H_∞-based system identification is given below, along with source and abstract for each paper. The *Notes* column contains a code, reflecting the applicability of the paper to the work in this study, and sometimes a brief note on the contents or usefulness of the given paper. The applicability code is a number (in [0,10]; 10 denotes greatest applicability) and a letter (**A** and **a** mean high applicability, and **D** or (**d**) little or no applicability; uppercase denotes that the paper comes from one of the principle authors in H_∞-based system identification: Gu, Khargonekar, Helmicki, Jacobson, Nett, Partington, or Mäkilä).

#	*Authors*	*Title*	*Source*	*Abstract*	*Notes*
1	V.M. Adamjan, D.Z. Arov, and M.G. Krein, 1971. "Analytic Properties of Schmidt Pairs for a Hankel Operator and the Generalized Schur-Takagi Problem." *Mathematics of the USSR – Sbornik*, **15**(1), Sept. 1971, 31-73 (Russian original Tom 86(128)).			This article is a study of infinite Hankel matrices and approximation problems connected with them.	9a
2	H. Akçay, G. Gu, and P.P. Khargonekar, 1992. "Identification in H_∞ with Nonuniformly Spaced Frequency Response Measurements." *1992 American Control Conference*, Chicago, Illinois, June 24-26, 1992. Proceedings (American Automatic Control Council, Evanston, Illinois), 246-250.			In this paper, the problem of "system identification in H_∞" is investigated in the case when the given frequency response data is not necessarily on a uniformly spaced grid of frequencies. A large class of robustly convergent identification algorithms are derived.	7B non-uniformly-spaced frequencies
3	H. Akçay, G. Gu, and P.P. Khargonekar, 1993. "A Class of Algorithms for Identification in H_∞: Continuous-Time Case." *IEEE Transactions on Automatic Control*, **38**(2), Feb. 1993, 289-294.			In this note, the problem of system identification in H_∞ for the continuous-time case is investigated. It is shown that the class of systems with a lower bound on the relative stability, an upper bound on the steady state gain, and an upper bound on the roll-off rate is *admissible*. This allows one to develop a class of robustly convergent nonlinear algorithms. The algorithms in this class have a two-stage structure, and are characterized by the use of window functions. Explicit worst-case error bounds in H_∞ norm between the identified model and the unknown system are given for a particular algorithm. Finally, an example is provided to illustrate the application of the results obtained.	7B continuous time
4	T. C. P. M. Backx and A. A. H. Damen	Identification for the Control of MIMO Industrial Processes.	*IEEE Transactions on Automatic Control*, **37**(7), July 1992, 980-986.	A procedure for the identification of industrial processes with the intention of control system design is proposed, discussed, and illustrated by an application to a full-scale production process. The various identification steps are motivated, keeping industrial applicability of the procedure in mind. The MIMO model set used is the common denominator form or minimum polynomial form. Parameter estimation is performed in several steps, thus adapting to estimation and control requirements. As an indicative example of practical results obtained, the identification and control of a quartz tube glass process is described.	0d
5	A. Bahri and A.J. Helmicki, 1995. "H_∞ Identification-Based Robust Control System Design." *1995 American Control Conference*, Seattle, Washington, June 21-23, 1995. Proceedings (American Automatic Control Council, Evanston, Illinois), 3556-3561.			In this paper the interaction between H_∞ identification and H_∞ robust control design problems is studied. An iterative solution for the coupled H_∞ identification and control problems is proposed, which involves pre-filtering the plant data. Some conditions on the pre-filter to permit convergence are derived. The use of pre-filtering is shown to significantly reduce the number of experiments required at successive iterations.	3C
6	E.-W. Bai and M.S. Andersland, 1994. "Stochastic and Worst Case System Identification Are Not Necessarily Incompatible." *Automatica*, **30**(9), Sept. 1994, 1491-1493.			Stochastic and worst case approaches to system identification are different and are usually treated separately. In this communique we investigate the effect that a projection operator has on the worst case behavior of estimates derived by stochastic identification algorithms. We show that under certain assumptions the projections of the stochastic estimates are convergent in the worst case setting. We illustrate this result by applying it to least squares and maximum likelihood algorithms.	6b relation between stochastic and worst-case ID
7	E.-W. Bai and S. Raman, 1994. "Robust System Identification with Noisy Experimental Data: Projection Operator and Linear Algorithms." *Automatica*, **30**(7), July 1994, 1203-1206.			In this paper we consider the problem of robust system identification with noisy time or frequency response measurement data. It is shown here that any linear identification algorithm which is convergent in the noise free case can be made robustly convergent in the presence of noise by incorporating a simple projection operator into the algorithm. The computation simplicity and faster rate of convergence distinguish this approach from other existing robustly convergent nonlinear identification techniques.	8a modification to linear algorithms to force robust noise convergence (uses *a priori* info)

#	Authors	Title	Source	Abstract	Notes
8	D. S. Bayard, Y. Yam, and E. Mettler	A Criterion for Joint Optimization of Identification and Robust Control.	*IEEE Transactions on Automatic Control*, 37(7), July 1992, 986-991.	A criterion for system identification is developed which is consistent with the intended use of the fitted model for modern robust control synthesis. Specifically, a joint optimization problem is posed which simultaneously determines the plant model estimate and control design, so as to optimize robust performance over the set of plants consistent with a specified experimental data set.	0d integrated ID/Control design
9	J. Chen, G. Gu, and C.N. Nett, 1993. "Worst Case Identification of Continuous Time Systems via Interpolation." *1993 American Control Conference*, San Francisco, California, June 2-4, 1993. Proceedings (American Automatic Control Council, Evanston, Illinois), 1544-1548.			We consider a worse case control oriented identification problem recently studied by several authors. This problem is one of the H_∞ identification in the continuous time setting. We give a less conservative formulation of this problem. The available a priori information consists of a lower bound on the relative stability of the plant, a *frequency dependent* upper bound on a certain gain associated with the plant, and an upper bound on the noise level. The available experimental information consists of a finite number of noisy plant point frequency response samples. The objective is to identify from the given a priori and experimental information an uncertain model that includes a stable nominal plant model and a bound on the modeling error measured in H_∞ norm. Our main contributions include both a new identification algorithm and several new *explicit* lower and upper bounds on the identification error. The algorithm proposed belongs to the class of *interpolatory algorithms* which are known to posses a desirable optimality property under a certain criterion. The error bounds presented improve upon the previously available ones in both the aspects of providing a more accurate estimate of the identification error as well as establishing a faster convergence rate for the proposed algorithm.	6B continuous time via interpolation of TFs in the frequency domain; uses *a priori* info; some good lit review info
10	J. Chen, G. Gu, and C.N. Nett, 1994. "Worst Case Identification of Continuous Time Systems via Interpolation." *Automatica*, 30(12), Dec. 1994, 1825-1837.			A suboptimal identification algorithm and several improved bounds for identification error are developed based upon the Nevanlinna-Pick interpolation procedure for a worst case H_∞ identification problem in the continuous time setting. We consider a worst case robust control oriented identification problem recently studied by several authors. This problem is one of H_∞ identification in the continuous time setting. We give a more general formulation of this problem. The available a priori information in this paper consists of a lower bound on the relative stability of the plant, a frequency dependent upper bound on a certain gain associated with the plant, and an upper bound on the noise level. The available experimental information consists of a finite number of noisy plant point frequency response samples. The objective is to identify, from the given a priori and experimental information, an uncertain model that includes a stable nominal plant model and a bound on the modeling error measured in H_∞ norm. Our main contributions include both a new identification algorithm and several new 'explicit' lower and upper bounds on the identification error. The proposed algorithm belongs to the class of 'interpolatory algorithms' which are known to posses a desirable optimality property under a certain criterion. The error bounds presented improve upon the previously available ones in the aspects of both providing a more accurate estimate of the identification error as well as establishing a faster convergence rate for the proposed algorithm.	6B continuous time via interpolation of TFs in the frequency domain
11	J. Chen, C. N. Nett, and M. K. H. Fan	Worst Case System Identification in H_∞: Validation of Apriori Information, Essentially Optimal Algorithms, and Error Bounds	*1992 American Control Conference*, Chicago, Illinois, June 24-26, 1992. Proceedings (American Automatic Control Council, Evanston, Illinois), 251-257.	This paper is concerned with a particular control-oriented system identification problem recently considered by several authors. This problem has been referred to as the problem of worst-case system identification in H_∞ in the literature. The formulation of this problem is worst-case/deterministic in nature. The available apriori information consists of a lower bound on the relative stability of the plant, an upper bound on a certain gain associated with the plant, and an upper bound on the noise level. The available aposteriori information consists of a finite number of noisy plant point frequency response samples. The objective is to identify the plant transfer function in H_∞ using the available apriori and aposteriori information. In this paper we resolve several important open issues pertaining to this problem. First, a method is presented for developing confidence that the available apriori information is correct. This method requires the solution of a certain non-differentiable convex programming problem. This algorithm is (worst-case strongly) optimal to within a factor of two. Finally, new upper and lower bounds on the optimal identification error for this problem are derived and used to estimate the identification error associated with the algorithm presented here. Interestingly, the development of each of the results described above draws heavily upon the classical Nevanlinna-Pick optimal interpolation theory. As such, the results of this paper establish a clear link between the areas of system identification and optimal interpolation theory.	5B

#	Authors	Title	Source	Abstract	Notes
12	J. Chen, C. N. Nett, and M. K. H. Fan	Optimal Non-Parametric System Identification From Arbitrary Corrupt Finite Time Series: A Control–Oriented Approach.	*1992 American Control Conference*, Chicago, Illinois, June 24-26, 1992. Proceedings (American Automatic Control Council, Evanston, Illinois), 279-285.	In this paper we formulate and solve a control-oriented system identification problem for single-input, single-output, linear, shift-invariant, distributed parameter plants. In this problem the available apriori information is minimal, consisting only of worst-case/deterministic, time dependent, upper and lower bounds on the plant impulse response and the additive output noise. The available aposteriori information consists of a corrupt finite output time series obtained in response to a known, non-zero but otherwise arbitrary, applied input. A novel system identification method is presented for this problem. This method maps the available apriori and aposteriori information into and "uncertain model" of the plant. The resulting uncertain plant model is comprised of a nominal plant model, a bounded additive output noise, and a bounded additive model uncertainty. The upper bound on the model uncertainty is explicit, worst-case/deterministic in nature, and expressed in terms of both the l_1 and H_∞system norms. Under the assumption that the available apriori information is "correct" for the underlying physical plant, the resulting uncertain plant model has the property that it not only "explains" the available aposteriori information, but will also explain all aposteriori information observed in the future. Because this property hinges on the correctness of the available apriori information, a method is also presented for developing confidence that the available apriori information is in fact correct. Both the method for building confidence in the correctness of the available apriori information and the method for identifying the uncertain plant model are quite simple computationally, requiring only the solution of a single linear programming problem. Nonetheless, these methods can be shown to have certain well-defined, physically meaningful optimality properties. These optimality properties make clear that several aspects of the methods can not be significantly improved upon. Finally, two special cases of the general methods which arise often in applications are considered in detail. In the first case the applied input is an impulse function, and in the second case the applied input is a step function. For these special cases the relevant linear programs are solved explicitly, and additional optimality results are established.	3C
13	J. Chen, C. N. Nett, and M. K. H. Fan	Worst Case System Identification in H_∞: Validation of *a Priori* Information, Essentially Optimal Algorithms, and Error Bounds	*IEEE Transactions on Automatic Control*, **40**(7), July 1995, 1260-1265.	In this paper we resolve several important open issues pertaining to a worst-case control-oriented system identification problem known as identification in H_∞. First, a method is presented for developing confidence that certain *a priori* information available for identification is not invalid. This method requires the solution of a certain nondifferentiable convex program. Second, an essentially optimal identification algorithm is constructed. This algorithm is (worst-case strongly) optimal to within a factor of two. Finally, new upper and lower bounds on the optimal identification error are derived and used to estimate the identification error associated with the given algorithm. Interestingly, the development of each of these results draws heavily upon the classical Nevanlinna-Pick interpolation theory. As such, our results establish a clear link between the areas of system identification and optimal interpolation theory. Both the formulation and techniques in this paper are applicable to problems where the frequency data available for identification may essentially be arbitrarily distributed.	5B develops some new (suboptimal) algorithms
14	J. Chen and S. Wang, 1995. "New Time-Domain Algorithms for H_∞ Identification." *1995 American Control Conference*, Seattle, Washington, June 21-23, 1995. Proceedings (American Automatic Control Council, Evanston, Illinois), 1976-1980.			We discuss several issues pertaining to a time-domain H_∞ problem. These issues are centered at an inherent trade-off between algorithm optimality and model as well as computational complexity. We provide a number of simple "nearly" interpolatory algorithms which may be employed to lessen somewhat the computational complexity and for constructing a lower order model.	3C
15	R. Y. Chiang, Y. Yam, E. Mettler, D. S. Bayard, A. Ahmed, and F. Y. Hadaegh	System Identification and H_∞ Synthesis for a Non-Collocated Space Structure Control Experiment	*1993 American Control Conference, San Francisco*, California, June 2-4, 1993. Proceedings (American Automatic Control Council, Evanston, Illinois), 3033-3037.	This paper documents a robust control design experiment in a technology demonstration for Advanced Reconfigurable Control (ARC). The objective of the experiment is to develop an integrated identification and robust synthesis methodology for vibration suppression of large space structures. The overall methodology has been partially implemented and evaluated on a complex flexible structure experiment at JPL. The identification approach is based on a recent frequency domain method which estimates both a state space model and an additive uncertainty weighting for robust control Design (Bayard and Yam, "Freq. domain ID for robust control design," *Modeling of Uncertainty in Control Systems*, Smith and Doyle, eds., Springer-Verlag, in press). The control part is based on a novel H_2/H_∞ approach with a hierarchical MIMO inner/outer loop design structure. This case study indicates that the integrated design methodology provides an effective approach to developing vibration controllers for large space structures, or related applications involving plants of commensurate complexity.	0d integrated ID/control; same testbed as analyzed with H_∞ ID by Friedman and Khargonekar (1995)
16	H. Dai and N. K. Sinha	Robust Identification of Systems Using Block-Pulse Functions.	*IEE Proceedings-D Control Theory and Applications*, **139**(3), May 1992, 308-316.	A robust method is employed to identify the unknown parameters of both linear and bilinear systems. Using block-pulse functions, this method expands the system input and output utilising an approach that minimises a robust criterion to reduce the effect of noise, especially large errors (called outliers) on the expansion coefficients. These coefficients are then used to obtain robust estimates of parameters. A Theorem showing convergence of this method is included. Simulation results provided in this paper demonstrate robustness and convergence of the proposed robust method. It can be concluded that this method is superior to the non-robust version in the presence of noise, especially outliers.	0d

#	Authors	Title	Source	Abstract	Notes
17	H. Dai and N. K. Sinha	A Robust Off-Line Output Error Method for System Identification.	*IEEE Transactions on Industrial Electronics*, 39(4), Aug. 1992, P285-292.	In this paper, the "model reference" technique and Huber's minimax principle have been successfully used to develop an off-line output error method for robust identification of systems. This method is named the robust iterative output error method with modified residuals. A convergence analysis of the proposed method has been included as well as some simulation results. In the presence of a small number of large errors (called outliers) in the input-output data, the presented method has demonstrated its distinctive advantages over not only the nonrobust methods but also previously developed robust methods. The main advantages are a fast convergence speed and satisfactory robustness. It can be concluded that the method developed in this paper is much superior to the other methods and therefore can be widely used in many real-time applications.	0d
18	G. Didinsky, Z. Pan, and T. Basar, 1995. "Parameter Identification for Uncertain Plants using H_∞ Methods." *Automatica*, 31(9), Sept. 1995, 1227-1250.			We demonstrate the effective use of H_∞ filtering and cost-to-come methods for parameter identification in (deterministic) uncertain plants that are linear in the unknown parameters, but nonlinear otherwise. The *cost-to-come* method is an approach that has been used earlier to solve linear and nonlinear H_∞ optimal control and filtering problems. It consists of constructing a *cost-to-come* function, which assists in the design of an 'optimal' observer scheme. The method is used here in the design of a parameter identification scheme for uncertain plants, where measurements on the state of the system are available, but not on its derivative. Two approaches are adopted, in both of which the parameter estimation problem is formulated as an H_∞ filtering problem. One of the approaches uses a more standard prefiltering of the past states, input and disturbance signals. The other approach is a novel design method, which leads to a new class of identification schemes. It involves two subproblems: FSDI (full-state-derivative information) problem, where it is assumed that both the state and its derivative are available to the parameter estimator, and NPFSI (noise-perturbed FSI) problem, where the estimator is assumed to measure a noise-perturbed measurement of the state. In the latter problem we use singular perturbation methods to prove asymptotic convergence of the performance of the identifier to that of the unperturbed case, thus providing an asymptotically optimal solution to the FSI (full-state measurement) problem. To illustrate both approaches, several simulation studies on a numerical example are provided.	6b requires at least full state feedback; notably different from other H_∞ ID schemes
19	B. Franke and P. Löhnberg	Optimal Identification Time for Control	*1993 American Control Conference*, San Francisco, California, June 2-4, 1993. Proceedings (American Automatic Control Council, Evanston, Illinois), 1549-1553	This paper deals with consecutive (open-loop) identification and (closed-loop) control of a linear, time-invariant SISO process. The partition of a fixed total time between identification and control is optimised according to and LQ criterion. For a static gain process, an analytical expression for the optimal identification time as a function of the *a priori* parameter mean and variance is derived. For an integrator process, the optimal identification time obtained from simulations is approximated by an analytical formula. Both procedures are applied on-line by replacing the *a priori* statistics by their estimates. Finding an analytical expression for higher order processes appeared infeasible. Therefore, for a static gain process, at each sampling instant, the predicted cost for continuing identification is compared to the predicted cost for starting control. This procedure is expected to be applicable for dynamical systems of high order.	0d
20	J.H. Friedman, 1996. "Identification, Modeling, and Control of Flexible Structures." Ph.D. dissertation, Department of Aerospace Engineering, University of Michigan, Ann Arbor, Michigan, 1996.			Two important components in control design are the model development prior to control design and the performance analysis after the design is complete. Model development generally falls into two categories: (1) first principles modeling based on the physics of the individual components of a system; and (2) system identification based on experimental data. The problem of performance analysis can be quite broad, ranging from the step response of a system to the computation of system norms. In this dissertation we address problems from both of these fields of research. The modeling work presented in this dissertation includes both first principles and experimental modeling. We develop a first principles model of the dynamics of an M1/M1A1 tank as a motivational example and a simulation testbed on which we demonstrate the identification and control analysis tools developed in this dissertation. In the field of identification there are a number of methods available to engineers. Among these methods, the algorithms for solving the problem of identification in H_∞ have received much attention recently. The focus of the attention has been to develop the theoretical properties of the algorithms; however, less attention has been paid to the engineering applications of the algorithms. It is this practical application which is the primary focus of our work in system identification. This dissertation includes results on the key issues in engineering applications of the two-stage nonlinear algorithms, a step-by-step recipe for the selection of the design parameters, and heuristic rules for successful applications. In the area of performance analysis, we examine the computation of the worst-case and average H_2 norm of a family of linear systems with constant real parametric uncertainty. It is shown that when the system matrices depend affinely on real uncertain parameters, any quadratic performance index will be a rational function of these parameters. Using this fact, in the case of a single real parameter, the computation of the worst-case H_2 norm is quite similar to the computation of the H_∞ norm of an auxiliary system and the average performance becomes the integral of a rational function. Several examples are included to illustrate the utility of these results.	10A

#	Authors	Title	Source	Abstract	Notes
21	J.H. Friedman and P.P. Khargonekar, 1995a. "A Comparative Applications Study of Frequency Domain Identification Techniques." *1995 American Control Conference*, Seattle, Washington, June 21-23, 1995. Proceedings (American Automatic Control Council, Evanston, Illinois), 3055-3059.			In this paper, we compare the results of the following frequency identification algorithms: Sanathanan and Koerner (SK) algorithm, nonlinear least squares via Levenberg-Marquardt method, and the two-stage nonlinear algorithm. We also present a recipe for the application of the two-stage nonlinear algorithm. The emphasis of this paper is on the application and comparison of the algorithms developed in the literature to three case studies.	10A good, but *very* brief, summary of the 2-stage nonlinear H_∞ ID method; brief examination of a few examples
22	J.H. Friedman and P.P. Khargonekar, 1995b. "Application of Identification in H_∞ to Lightly Damped Systems: two case studies." *IEEE Transactions on Control Systems Technology*, 3(3), Sept. 1995, 279-289.			This paper presents an approach to the frequency domain identification of lightly damped systems. It is based on the recent work in the area of identification in H_∞. The emphasis of this paper is on the application of the algorithms developed in the literature to two case studies. Results show that the algorithms for identification in H_∞ are capable of producing good models for highly flexible systems.	10A excellent example on real structures
23	G. C. Goodwin, M. Gevers, and B. Ninness	Quantifying the Error in Estimated Transfer Functions with Application to Model Order Selection.	*IEEE Transactions on Automatic Control*, 37(7), July 1992, 913-928.	Previous results on estimating errors or error bounds on identified transfer functions have relied upon prior assumptions about the noise and the unmodeled dynamics. This prior information took the form of parameterized bounding functions or parameterized probability density functions, in the time or frequency domain with known parameters. Here we show that the parameters that quantify this prior information can themselves be estimated from the data using a maximum likelihood technique. This significantly reduces the prior information required to estimate transfer function error bounds. We illustrate the usefulness of the method with a number of simulation examples. The paper concludes by showing how the obtained error bounds can be used for intelligent model order selection that takes into account both measurement noise and under-modeling. Another simulation study compares our method to Akaike's well-known FPE and AIC criteria.	0d
24	G. Gu	Suboptimal Algorithms for Worst Case Identification in H_∞ and Model Validation	*IEEE Transactions on Automatic Control*, 39(8), Aug. 1994, 1657-1661.	New algorithms based on convex programming are proposed for worst case system identification. The algorithms are optimal with a factor of two asymptotically. Further, model validation, or data consistency, is embedded in the identification process. Explicit worst case identification error bounds in the H_∞ norm are also derived for both uniformly and nonuniformly spaced frequency response samples.	5B more suboptimal algorithms
25	G. Gu, C.-C. Chu, and G. Kim, 1994. "Linear Algorithms for Worst Case Identification in H_∞ with Applications to Flexible Structures." *1994 American Control Conference*, Baltimore, Maryland, June 29 - July 1, 1994. Proceedings (American Automatic Control Council, Evanston, Illinois), 112-116.			This paper is concerned with linear algorithms for identification in H_∞ which have been studied in (Helmicki, Jacobson, and Nett, 1993. "Identification in H_∞: linear algorithms." *IEEE Transactions on Automatic Control*, **38**, May 1993, 819-826). It is shown that the two different linear algorithms in (ibid.) can be unified into a single one which can be further extended to nonuniformly spaced frequency response samples with exponential convergence for the noise free case. Improved upper bounds for the corresponding identification errors are derived. Applications to the identification of lightly damped systems such as flexible structures are also considered.	6B
26	G. Gu and P.P. Khargonekar, 1991. "Linear and Nonlinear Algorithms for Identification in H_∞ with Error Bounds." *1991 American Control Conference*, Boston, Massachusetts, June 26-28, 1991. Proceedings (American Automatic Control Council, Green Valley, Arizona), 64-69.			In this paper, a linear and a nonlinear algorithm are presented for the problem of system "identification in H_∞," posed by Helmicki, Jacobson, and Nett. We derive some error bounds for the linear algorithm which indicate that if the model error is not too high, then this algorithm has good guaranteed error properties. The linear algorithm requires only FFT (fast Fourier transform) computations. A nonlinear algorithm, which requires an additional step of solving a Nehari best approximation problem, is also presented that has the robust convergence property.	10A
27	G. Gu and P.P. Khargonekar, 1992a. "A Class of Algorithms for Identification in H_∞." *Automatica*, **28**(2), March 1992, 299-312.			In this paper, a class of algorithms for the problem of system identification in H_∞ are investigated. These algorithms are characterized by a two-stage structure and involve a class of window functions. Some conditions in terms of properties of the window functions are derived, which guarantee robust convergence of the algorithms. Identification errors are analyzed for several common window functions. This leads to some insights into the trade-off between the error induced by approximation and that due to noise.	10A great explanation of basic H_∞ ID algorithm; good analytical example
28	G. Gu and P.P. Khargonekar, 1992b. "Linear and Nonlinear Algorithms for Identification in H_∞ with Error Bounds." *IEEE Transactions on Automatic Control*, 37(7), July 1992, 953-963.			In this paper, a linear algorithm and a nonlinear algorithm are presented for the problem of "system identification in H_∞," posed by Helmicki, Jacobson, and Nett for discrete-time systems. We derive some error bounds for the linear algorithm which indicate that it is not robustly convergent. However, the worst-case identification error is shown to grow as $\log(n)$ where n is the model order. A new robustly convergent nonlinear algorithm is derived, and bounds on the worst-case identification error (in the H_∞ norm) are obtained.	10A good explanation of basic H_∞ ID algorithm; good analytical example
29	G. Gu and P.P. Khargonekar, 1993. "Frequency Domain Identification of Lightly Damped Systems: The JPL Example." *1993 American Control Conference*, San Francisco, California, June 2-4, 1993. Proceedings (American Automatic Control Council, Evanston, Illinois), 3052-3056.			This paper describes application of the recent work on "identification in H_∞" to the data for a JPL flexible space structure.	10A practical example

#	Authors	Title	Source	Abstract	Notes
30	G. Gu, P.P. Khargonekar, and E.B. Lee, 1989. "Approximation of Infinite Dimensional Systems." *IEEE Transactions on Automatic Control*, 34(6), June 1989, 610-618.			Approximation of infinite-dimensional system models was studied using a Fourier transform technique. Convergence conditions were established and a frequency response error bound in terms of the H_∞ norm derived. The approximate model can be directly computed using an FFT type algorithm. Examples illustrate the method.	0C
31	G. Gu and P. Misra.	Identification of Linear Time-Invariant Systems From Frequency-Response Data Corrupted By Bounded Noise.	*IEE Proceedings-D Control Theory and Applications*, 139(2), March 1992, 135-140.	A unified approach is developed for identification of linear time-invariant systems. It is shown that, given the experimental frequency-response data of the system, the plant can be identified using a simple, numerically reliable algorithm. Further, an error bound is derived for exponentially stable systems when the frequency-response data are corrupted by bounded noise. An example is presented to illustrate the proposed algorithm.	5B studies the divergence rate of linear H_∞ ID algorithms with noisy data
32	A.J. Helmicki, C.A. Jacobson, and C.N. Nett, 1989. "H_∞ Identification of Stable LSI Systems: A Scheme with Direct Application to Controller Design." *1989 American Control Conference*, Pittsburgh, Pennsylvania, June 21-23, 1989. Proceedings (American Automatic Control Council, Green Valley, Arizona), 1428-1434.			In this paper several techniques are given for the identification of stable LSI discrete time systems from input-output data. Explicit H_∞ norm error bounds are given and convergence in the noise free and the uniformly bounded deterministic noise case are established. The assumptions made on the unknown system are minimal and are limited throughout the paper to a lower bound on the decay rate of the unknown system and an upper bound on the gain of the unknown system. Given this information an experiment and a construction are specified: the experiment involves obtaining a specified number of frequency measurements of the unknown systems at a set of specified frequencies; the construction uses this experimental data to generate an identified model with prescribed H_∞ norm error tolerance to the unknown system. The resulting model identification process is highly efficient from a computational point of view.	10A one of the earliest papers on H_∞ ID
33	A.J. Helmicki, C. A. Jacobson, and C.N. Nett, 1990a. "Identification in H_∞: A Robustly Convergent, Nonlinear Algorithm." *1990 American Control Conference*, San Diego, California, May 23-25, 1990. Proceedings (American Automatic Control Council, Green Valley, Arizona), 386-391.			In this paper a system identification technique is developed which is compatible with current robust controller design methodologies. This technique is applicable to a broad class of stable, distributed, linear, shift-invariant systems. The information necessary for the application of this technique consists of a priori estimates on the relative stability and "steady state" gain of the unknown system together with a finite number of possibly corrupt frequency response estimates. Given this information an algorithm is specified which yields both an identified model and explicit H_∞ norm error bounds. Several interesting properties of this algorithm are also discussed. Among them, the fact the algorithm is a nonlinear function of the frequency response data, and that it is robustly convergent with respect to the a prior information on relative stability and gain are singled out as characteristics which distinguish this algorithm from other currently under development by the authors.	10A
34	A.J. Helmicki, C.A. Jacobson, and C.N. Nett, 1990b. "Identification in H_∞: The Continuous-Time Case." *1990 American Control Conference*, San Diego, California, May 23-25, 1990. Proceedings (American Automatic Control Council, Green Valley, Arizona), 1893-1898.				7B
35	A.J. Helmicki, C.A. Jacobson, and C.N. Nett, 1990c. "Identification in H_∞: Linear Algorithms." *1990 American Control Conference*, San Diego, California, May 23-25, 1990. Proceedings (American Automatic Control Council, Green Valley, Arizona), 2418-2423.			In this paper a series of system identification techniques are developed which are compatible with current robust controller design methodologies. These techniques are applicable to a broad class of stable, distributed, linear, shift-invariant systems. The information necessary for their application consists of a priori estimates on the relative stability and "steady state gain" of the unknown system together with a finite number of possibly corrupt frequency response samples. Given this information the algorithms established yield both identified models and explicit H_∞ norm error bounds. These algorithms are developed as extensions of a recently proposed polynomial interpolation approach to H_∞ identification which is shown here to diverge in the face of corrupted data. The fact that these algorithms are linear functions of the frequency response data and depend explicitly on the a priori information are singled out as characteristics which distinguish them from other algorithms recently established by the authors.	10A

#	Authors	Title	Source	Abstract	Notes
36	A.J. Helmicki, C.A. Jacobson, and C.N. Nett, 1991a. "Fundamentals of Control-Oriented System Identification and Their Application for Identification in H_∞." *1991 American Control Conference*, Boston, Massachusetts, June 26-28, 1991. Proceedings (American Automatic Control Council, Green Valley, Arizona), 89-99.			This paper examines the system identification problem from the standpoint of control system design. Noting first that nearly all robust control design methods require explicit worst-case/deterministic bounds on the existing plant uncertainty, it is argued that the class of system identification methods which are inherently compatible with robust control design methods — or control-oriented — is a subset of the class of system identification methods which yield and explicit worst-case/deterministic bound on the resulting identification error. An abstract theoretical framework for control-oriented system identification is then developed. This framework is inherently worst-case/deterministic in nature, and makes precise such notions as identification error, algorithm convergence, and algorithm optimality from a worst-case/deterministic standpoint. Finally, the abstract theoretical framework is utilized to formulate and solve two related control-oriented system identification problems for stable, linear shift invariant, distributed parameter plants. In each of these problems the assumed a priori information is minimal, consisting only of a lower bound on the relative stability of the plant, an upper bound on a certain gain associated with the plant, and an upper bound on the noise level. In neither case are any assumptions made concerning the structure of either the plant (i.e., dynamic order, relative order, etc.) or the noise (i.e., zero-mean, etc.). The first of these problems involves identification of a point sample of the plant frequency response from a noisy, finite, output time series obtained in response to an applied sinusoidal input with frequency corresponding to the frequency point of interest. This problem leads naturally to the second problem, which involves identification of the plant transfer function in H_∞ from a finite number of noisy point samples of the plant frequency response. Robust convergent, (essentially) asymptotically optimal plans of identification algorithms are provided for each of these two problems. The plans provided for the second problem yield and explicit worst-case/deterministic bound on the H_∞-norm of the resulting identification error at each step of the plan. As such, the identification methods obtained by combining the given plans for the two problems are well-suited for use in conjunction with currently popular H_∞ robust control design methods, and hence may be regarded as being inherently control-oriented.	10A excellent for background and lit. review
37	A.J. Helmicki, C.A. Jacobson, and C.N. Nett, 1991b. "Control Oriented System Identification: A Worst-Case Deterministic Approach in H_∞." *IEEE Transactions on Automatic Control*, **36**(10), Oct. 1991, 1163-1176.			In this paper we formulate and solve two related control-oriented system identification problems for stable linear shift-invariant distributed parameter plants. In each of these problems the assumed *a priori* information is minimal, consisting only of a lower bound on the relative stability of the plant, an upper bound on a certain gain associated with the plant, and and upper bound on the noise level. In neither case are any assumptions made concerning the structure of either the plant (i.e., dynamic order, relative order, etc.) or the noise (i.e., zero-mean, etc.). The first of these problems involves identification of a point sample of the plant frequency response from a noisy finite output time series obtained in response to an applied sinusoidal input with frequency corresponding to the frequency point of interest. This problem leads naturally to the second problem, which involves identification of the plant transfer function in H_∞ from a finite number of noisy point samples of the plant frequency response. Concrete plans of identification algorithms are provided for each of these two problems. Explicit worst-case/deterministic error bounds are provided for each algorithm in these plans. These bounds establish that the given plans of algorithms are robustly convergent and (essentially) asymptotically optimal. Additionally, these bounds provide an *a priori* computable H_∞ uncertainty specification, corresponding to the resulting identified plant transfer function, as an explicit function of the plant *a priori* information, noise a priori information, and experiment duration. As such, the approach to system identification developed in this paper is well-suited for use in conjunction with currently popular H_∞ robust control design methods, and for this reason may be regarded as being inherently control-oriented.	10A good intro and summary; confusing derivation of linear algorithm with *a priori* information requirement
38	A.J. Helmicki, C.A. Jacobson, and C.N. Nett, 1992. "Worst-Case Deterministic Identification in H_∞: The Continuous-Time Case." *IEEE Transactions on Automatic Control*, **37**(5), May 1992, 604-610.			In this note, recent results obtained by the authors for worst-case/deterministic H_∞ identification of discrete-time plants are extended to continuous-time plants. The problem considered involves identification of the transfer function of a stable strictly proper continuous-time plant from a finite number of noisy point samples of the plant frequency response. The assumed *a priori* information consists of a lower bound on the relative stability of the plant, an upper bound on a certain gain associated with the plant, an upper bound on the "roll-off rate" of the plant, and an upper bound on the noise level. Concrete plans of identification algorithms are provided for this problem. Explicit worst-case/deterministic error bounds are provided for each algorithm in these plans. These bounds establish that the given plans of algorithms are robustly convergent and (essentially) asymptotically optimal. Additionally, these bounds provide an *a priori* computable H_∞ uncertainty specification, corresponding to the resulting identified plant transfer function, as an explicit function of the plant and noise *a priori* information and the data cardinality.	7B continuous-time; some good intro material

#	Authors	Title	Source	Abstract	Notes
39	A. J. Helmicki, C. A. Jacobson, and C. N. Nett.	Least Squares Methods for H_∞ Control-Oriented System Identification.	*1992 American Control Conference*, Chicago, Illinois, June 24-26, 1992. Proceedings (American Automatic Control Council, Evanston, Illinois), 258-264.	This paper presents a series of system identification algorithms that yield identified models which are compatible with current robust controller design methodologies. These algorithms are applicable to a broad class of stable, distributed, linear, shift-invariant plants. The a priori information necessary for their application consists of a lower bound on the relative stability of the unknown plant, an upper bound on a certain gain associated with the unknown plant, and an upper bound on the noise level. The a posteriori data information consists of a finite number of noise point frequency response estimates of the unknown plant. The specific contributions of this paper are to examine the extent to which certain standard Hilbert space or least squares methods are applicable to the H_∞ system identification problem considered. Results are established that connect the H_2 error of the least square methods to the H_∞ error needed for control-oriented system identification. In addition, the notion of a posteriori error bounds is introduced and used to establish sequentially optimal or adaptive algorithms based on these Hilbert space approaches.	6B
40	A. J. Helmicki, C. A. Jacobson, and C. N. Nett	Least Squares Methods for H_∞ Control-Oriented System Identification.	*IEEE Transactions on Automatic Control*, **38**(5), May 1993, 819-826.	This note presents a series of system identification algorithms that yield identified models which are compatible with current robust controller design methodologies. These algorithms are applicable to a broad class of stable, distributed, linear, shift-invariant plants. The *a priori* information necessary for their application consists of a lower bound on the relative stability of the unknown plant, an upper bound on a certain gain associated with the unknown plant, and an upper bound on the noise level. The *a posteriori* data information consists of a finite number of corrupted point frequency response estimates of the unknown plant. The specific contributions of this note are to examine the extent to which certain standard Hilbert space or least squares methods are applicable to the H_∞ system identification problem considered. Results are established that connect the H_2 error of the least squares methods to the H_∞ error needed for control-oriented system identification.	5B
41	H. Hjalmarsson and L. Ljung	Estimating Model Variance in the Case of Undermodeling.	*IEEE Transactions on Automatic Control*, **37**(7), July 1992, 1004-1008.	A reliable quality estimate of a given model is a prerequisite for any reasonable use of the model. The model error consists of two different contributions: the *bias error* and the *random error*. In this contribution, we show how the size (variance) of the random error can be reliably estimated in the case where a true system description cannot be achieved in the model structure used. This consistent error estimate can differ considerably from the conventionally used variance estimates which thus, could be quite misleading.	0d
42	C.A. Jacobson and G. Tadmor, 1993. "A Note on H_∞ System Identification With Probabilistic Apriori Information." *1993 American Control Conference*, San Francisco, California, June 2-4, 1993. Proceedings (American Automatic Control Council, Evanston, Illinois), 1539-1543.			This paper presents an analysis of H_∞ system identification where the apriori information given on the unknown system to be identified is described probabilistically. The H_∞ system identification problem concerns the construction of a linear shift invariant exponentially stable system from a combination of apriori and experimental information. The goal is to construct both a nominal system and an explicit quantification of model uncertainty in the H_∞ norm utilizing the apriori and experimental information. The experimental information assumed available in this paper is a set of corrupted point frequency estimates of the unknown system. The apriori information consists of a probability measure specifying the probability of bounds on the norm of the derivative of the unknown system. The problem formulation is given for this probabilistic setting with the error criterion allowing a probabilistic tolerance of identification to be given. It is shown that the probabilistic H_∞ problem is equivalent to a worst case problem that is constructed from the probabilistic one. This construction allows near optimal algorithms to be constructed for the probabilistic H_∞ identification problem.	7B combines probabilistic *a priori* bounds with H_∞ ID
43	I. Kollár	On Frequency-Domain Identification of Linear Systems.	*IEEE Transactions on Instrumentation and Measurement*, **42**(1), Feb. 1993, 2-6.	The maximum-likelihood estimation of the parameters of linear systems and the properties of the estimator (Estimator for Linear Systems, ELiS) have been described in a paper by R. Pintelon and J. Schoukens ("Robust identification of transfer functions in s- and z-domain," *IEEE Transactions on Instrumentation and Measurement*, **39**, Aug. 1990, 565-573). The mathematics used in the development of the method and the proofs are rather involved. However, several statements can be understood in heuristic terms. This paper discusses the complex-domain description of the method, which results in much simpler expressions. The method is also compared to other formations, giving more insight into the properties of the estimate. It turns out that robustness is at least partly due to the least-squares formulation. Derivations are avoided where possible, and intuitive explanations are given instead.	0d complex-domain description of the maximum-likelihood ID method
44	R.L. Kosut, G.C. Goodwin, and M.P. Polis, 1992. "Introduction, Special Issue on System Identification for Robust Control Design." *IEEE Transactions on Automatic Control*, **37**(7), July 1992, 899.			Introduction to the special issue.	2c some overall comments
45	R. L. Kosut, M. K. Lau, and S. P. Boyd	Set-Membership Identification of Systems with Parametric and Nonparametric Uncertainty.	*IEEE Transactions on Automatic Control*, **37**(7), July 1992, 929-941.	A method is presented for parameter set estimation where the system model is assumed to contain both parametric and nonparametric uncertainty. In the disturbance-free case, the parameter set estimate is guaranteed to contain the parameter set of the true plant. In the presence of stochastic disturbances, the parameter set estimate obtained from finite data records is shown to have the property that it contains the true-plant parameter set with probability one as the data length tends to infinity.	0d

#	Authors	Title	Source	Abstract	Notes
46	J. M. Krause and P. P. Khargonekar	A Comparison of Classical Stochastic Estimation and Deterministic Robust Estimation.	*IEEE Transactions on Automatic Control*, 37(7), July 1992, 994-1000.	This note compares the formulation and solution of two linear parameter estimation problems. The basic distinction in the problem formulations is the nature of the uncertainty. In one case, the uncertainty is generated by white Gaussian noise, and the solution is the Kalman filter. In the other case, the uncertainty is unmodeled dynamics in the unit ball in H_∞ or its nonlinear cover, and the particular solution studied here is a deterministic robust estimator which was introduced circa 1987. This note examines certain parallels between classical stochastic estimation (Kalman filtering) and the deterministic robust estimation. The similarities and differences are discussed in geometric terms, in philosophical terms, and in terms of the estimator's recursive implementation.	5B
47	P.M. Mäkilä, 1991a. "Laguerre Methods and H_∞ Identification of Continuous-Time Systems." *International Journal of Control*, 53(3), March 1991, 689-707.			H_∞ identification of stable continuous-time systems is studied using generalized Laguerre series methods. The theoretical basis of generalized Laguerre series methods in H_∞ identification is established by giving several results on frequency-unweighted and frequency-weighted approximations of different classes of infinite dimensional systems. An H_∞ identification technique based on step response data and Laguerre methods is given and analysed. Generalized Laguerre series methods are shown to provide H_∞ identification techniques which allow for frequency weighting. Furthermore, it is demonstrated that the theory of generalized Laguerre polynomials solves certain approximation problems in an analytical fashion for a class of delay systems.	8B good intro and comments on continuous-time H_∞ ID
48	P. M. Mäkilä	On Identification of Stable Systems and Optimal Approximation.	*Automatica*, 27(4), July 1991, 663-676.	Approximative modelling of stable continuous-time, possibly infinite dimensional, systems is studied based on an optimal approximation approach. Both approximation of analytical system representations (system approximation) as well as approximation of input-output data based system estimates (system identification) are considered. While special emphasis is given to approximative modelling in the H_∞ and Hankel norms, the l_1 and l_2 norm cases are also discussed. The model sets considered here are finite dimensional systems and time shift systems (simple delay systems). The theory of approximation numbers is shown to provide a convenient tool to study problems of identification of stable continuous-time systems in a deterministic framework with close connections to complexity considerations. Laguerre-Fourier series methods and Hankel operator techniques can be utilized to develop fully practical identification methods for continuous-time, possibly infinite dimensional, systems.	7B
49	P. M. Mäkilä	Identification of Stabilizable Systems: Closed-Loop Approximation.	*International Journal of Control*, 54(3), Sept. 1991, 577-592.	Approximate modelling and identification of linear shift-invariant, possibly unstable, discrete-time systems, or plants, is studied in a framework compatible with the so-called robust stability concept for feedback systems. This unified framework is based on approximate modelling of the plant in the gap and graph metrics which is achieved here through approximation of certain closed-loop transfer functions by finite-dimensional systems. Properties of this approximate inverse modelling approach are studied and concrete rate of approximation results are given. Furthermore, a consistency result in the gap and graph metrics is given for a certain experimental estimate of the plant constructed from closed-loop input-output data in a stationary noise set-up under mild conditions on the unknown plant.	0C
50	P.M. Mäkilä, 1991b. "Robust Identification and Galois Sequences." *International Journal of Control*, 54(5), Nov. 1991, 1189-1200.			Worst-case l_1 identification is studied for BIBO stable linear shift-invariant systems. It is shown that the Chebyshev identification method when used with Galois input designs satisfies a certain robust convergence property and provides l_1 model error bounds in worst-case identification of BIBO stable systems with a uniformly bounded noise set-up. The robust identification methodology developed is compatible with the modelling requirements of modern robust control design.	4B l_1 identification
51	P.M. Mäkilä, 1992. "Worst-Case Input-Output Identification." *International Journal of Control*, 56(3), Sept. 1992, 673-689.			We consider worst-case l_1 identification of causal linear shift-invariant systems from time series. Many results are given on general aspects of identification algorithm performance, existence of optimal algorithms, robust convergence, and input (experiment) design. The identification methodology studied here is compatible with the modelling requirements of modern robust control design.	3C l_1 identification
52	P.M. Mäkilä, 1993. "Robust Approximate Modeling from Noisy Point Evaluations." *1993 American Control Conference*, San Francisco, California, June 2-4, 1993. Proceedings (American Automatic Control Council, Evanston, Illinois), 1554-1560.			We consider approximate modeling of stable linear shift-invariant systems in the H_∞ sense from approximate point evaluations at approximately known frequencies. Two error structures for the point evaluations are studied: pointwise bounded error and a certain error averaging structure. A main motivation for the present work comes from currently active research problems concerning modeling for robust control design from experimental data. Several results are given on various aspects of approximation algorithm performance, and on robust convergence. A constrained least absolute deviations method based on minimizing the value of the error averaging prior subject to a modeling prior restricting the complexity of the behaviour of the model is proposed. This linear programming method is a strongly optimal algorithm within factor two with respect to the model and error priors used in its construction. Relationships between problems of identification of nominal models and uncertainty modeling are studied.	5B helpful lit review section.

#	Authors	Title	Source	Abstract	Notes
53	P.M. Mäkilä and J.R. Partington, 1991. "Robust Approximation and Identification in H_∞." *1991 American Control Conference*, Boston, Massachusetts, June 26-28, 1991. Proceedings (American Automatic Control Council, Green Valley, Arizona), 70-76.			Robust Approximation and identification of stable shift-invariant systems is studied in the H_∞ sense using a stable perturbation set-up. Issues of model set selection are addressed using the n-width concept: a concrete result establishes a priori knowledge for which a certain rational model set is optimal in the n-width sense. A general construction of interest to identification theory using ϵ-nets provides near-optimal identification methods tuned to the a priori knowledge about the system. A notion of robust convergence is defined so that any untuned identification method satisfying it has a generic well-posedness property for systems in the disk algebra. The existence of robustly convergent identification methods based on any complete model set in the disk algebra is established. It is also shown that the classical Fejér and de la Vallée-Poussin polynomial approximation operators provide robustly convergent identification methods. Furthermore, a result is given for optimal Hankel norm model reduction from experimentally obtained models.	7B
54	P.M. Mäkilä and J.R. Partington, 1992a. "Worst-Case Identification from Closed-Loop Time Series." *1992 American Control Conference*, Chicago, Illinois, June 24-26, 1992. Proceedings (American Automatic Control Council, Evanston, Illinois), 301-306.			This paper studies identification of linear shift-invariant systems from closed-loop time series. Identification (or modelling) error is measured here by distance functions which lead to the weakest convergence notions for systems such that closed-loop stability, in the sense of BIBO stability, is a robust property. Thus the identification methodology developed here is compatible with the requirements of robust control design under l_∞-stable coprime factor uncertainty. Worst-case identification error bounds in several distance functions are included.	6B
55	P.M. Mäkilä and J.R. Partington, 1992. "Robust Identification of Strongly Stabilizable Systems." *IEEE Transactions on Automatic Control*, **37**(11), Nov. 1992, 1709-1716.			For strongly stabilizable systems for which a strongly stabilizing controller is known approximately, we consider system identification in the graph, gap, and chordal metrics using robust H_∞ identification of the closed-loop transfer function in the framework proposed by Helmicki, Jacobson, and Nett. Error bounds are derived showing that robust convergence is guaranteed and that the identification can be satisfactorily combined with a model reduction step. Two notions of robust identification of stable systems are compared, and an alternative robust identification technique based on smoothing, which can be used to yield polynomial models directly, is developed.	7B some good intro material
56	P. M. Mäkilä and J. R. Partington	Robust Stabilization — BIBO Stability, Distance Notions and Robustness Optimization.	*Automatica*, **29**(3), May 1993, 681-693.	This paper studies robust stabilization of both linear shift-invariant causal systems in an l_p setting and linear time-invariant causal continuous-time systems in an L^p ($p = 1$ or ∞) setting. Two key technical results in the paper establish the existence of l_p and L^p stable normalized coprime factorizations for discrete-time and continuous-time systems, respectively, which have coprime factorizations as l_p and L^p stable operators. Several distance measures for systems are then introduced including the graph metric, the ρ function, the gap between the graphs of the systems, and the projection gap. It is shown that these distance measures lead to the weakest convergence notions for systems for which closed-loop stability is a robust property. The ρ function can be computed using the Dahleh-Pearson theory for l_1 (L^1) optimal control. Robustness optimization in a directed $\hat{p}$ function is shown to be closely related to robustness optimization for BIBO stable normalized coprime factor perturbations. This result connects the stability margin of Dahleh for coprime factor perturbations to the ρ function. These considerations are further supported by a robustness result in terms of the projection gap.	0C
57	P.M. Mäkilä and J.R. Partington, 1993. "Robust Approximate Modelling of Stable Linear Systems." *International Journal of Control*, **58**(3), Sept. 1993, 665-683.			Robust approximation and worst-case approximate modelling of stable shift-invariant systems from corrupted transfer function estimates are studied in the H_∞ sense. Connections between the problem formulations of the present work and certain problems of worst-case system identification, notably the Helmicki-Jacobson-Nett problem formulation for identification in H_∞, are established. Issues of model set selection are addressed using the n-width concept: a concrete result establishes *a priori* knowledge for which a certain rational model set is optimal in the n-width sense. A notion of robust convergence is defined so that any untuned approximation method satisfying it has a generic well-posedness property for systems in the disk algebra. The existence of robustly convergent approximation methods based on any complete model set in the disk algebra is shown in a constructive way. A framework is given in which approximate models can be obtained as stable perturbations of the true system: these can be combined with the classical Fejér and de la Vallée-Poussin polynomial approximation operators to provide robustly convergent approximation methods. Furthermore, concrete results are given for the fundamental problem of model reduction from corrupted transfer function estimates or from experimentally obtained models for the optimal Hankel norm approximation method and for a least squares method.	6B
58	P.M. Mäkilä, J.R. Partington, and T. K. Gustafsson, 1995. "Worst-case Control-relevant Identification." *Automatica*, **31**(12), Dec. 1995, 1799-1819.			During the past five years or so, several new research topics have emerged around issues of modelling of systems from data for the purpose of robust control design. These new topics include, among other things, identification in H_∞, identification in l_1, and model validation of uncertainty models. This paper introduces the reader to several recent developments in worst-case identification motivated by various issues of modelling of systems from data for the purpose of robust control design. Many aspects of identification in H_∞ and l_1 are covered including algorithms, convergence and divergence results, worst-case estimation of uncertainty models, model validation and control relevancy issues.	9A excellent, though very brief, summary of H_∞ and l_1 identification

#	Authors	Title	Source	Abstract	Notes
59	M. A. Mendlovitz	l_1-Optimal Estimation for Discrete-Time Linear Systems.	*IEEE Transactions on Signal Processing*, **41**(3), March 1993, 1103-1113.	A linear multichannel estimation problem with discrete-time linear shift-invariant models is formulated in the time domain as a minimum l_1 norm approximation problem. It is shown, using some key results from optimization theory, that solving the approximation problem is equivalent to solving a sequence of linear programming problems which terminates when an optimal or near-optimal solution is reached. The motivation for considering an l_1-optimal design versus l_2 or H_∞-optimal designs is presented. An example problem is solved to illustrate the computational procedure as well as to provide an opportunity to compare the relative performances of the l_1, l_2, and H_∞-optimal estimators in a practical situation.	0d
60	K.M. Nagpal and P.P. Khargonekar, 1991. "Filtering and Smoothing in an H_∞ Setting," *IEEE Transactions on Automatic Control*, **36**(2), Feb. 1991, 152-166.			In this paper we consider the problems of filtering and smoothing for linear systems in an H-infinity setting, i.e., the plant and measurement noises have bounded energies (are in L2), but are otherwise arbitrary. Two distinct situations for the initial condition of the system are considered: in one case the initial condition is assumed known, while in the other case, the initial condition is not known but the initial condition, the plant, and measurement noise are in some weighted ball of R(n) x L2. Both finite-horizon and infinite-horizon cases are considered. We present necessary and sufficient conditions for the existence of estimators (both filters and smoothers) that achieve a prescribed performance bound, develop algorithms that result in performance within the bounds. In case of smoothers, we also present the optimal smoother. The approach uses basic quadratic optimization theory in time-domain setting, as a consequence of which both linear time-varying and time-invariant systems can be considered with equal ease. (In the smoothing problem, for linear time-varying systems, we consider only the finite-horizon case).	
61	Z. Nehari, 1957. "On Bounded Bilinear Forms." *Annals of Mathematics*, **65**(1), Jan. 1957, 153-162.			*(no abstract)*	9a
62	P.J. Parker and R.R. Bitmead, 1987a. "Adaptive Frequency Response Identification." *26th IEEE Conference on Decision and Control*, Los Angeles, California, December 9-11, 1987. Proceedings (IEEE, New York), 348-353.			Given a stable, discrete time, single input single output system $G(z)$, but with only the input signal and the noise corrupted output signal available for measurement, we seek to find an approximation $G(z)$ — a finite impulse response (FIR) filter — with $\|G-\hat{G}\|_\infty = \sup |G(e^{j\theta})-\hat{G}(e^{j\theta})|$, $\theta\in(-\pi,\pi]$ bounded and small. The infinity norm has application in control theory and signal processing; furthermore, it is a measure of the deviation in frequency response between G and $\hat{G}$. Several previous papers, attempt to identify $G(z)$ in the frequency domain; these papers fail to bound $G-\hat{G}$ in any norm. Central to our method of identification is interpolation. First one estimates accurately $G(z)$ at n equally spaced frequencies. Here, n is a design parameter one may freely choose. This estimation relies on filtering the input and output signals appropriately. Then estimates of $G(e^{j2\pi k/n})$ come from a bank of $n/2$ decoupled least mean squares algorithms, each of two parameters; $\hat{G}(z)$ is then the unique FIR filter of degree $n-1$ with transfer function interpolating to these estimates. $\hat{G}(z)$ is computationally easy to evaluate. The resulting error bound has the form $\|G-\hat{G}\|_\infty \le MR^n + K(1+\log_2 n)$. Here M and R are constants, dependent on $G(z)$ with $R<1$; the accuracy of estimating $G(z)$ at the interpolation points determines K	4b
63	P.J. Parker and R.R. Bitmead, 1987b. "Approximation of Stable and Unstable Systems via Frequency Response Identification." *10th IFAC World Congress*, Munich, Germany, July 27-31, 1987. Proceedings (R. Isermann, ed., IFAC/Perfamon, Oxford, England, 1988), 358-363.				
64	J.R. Partington, 1991. "Robust Identification and Interpolation in H_∞." *International Journal of Control*, **54**(5), Nov. 1991, 1281-1290.			We consider system identification in H_∞ in the framework proposed by Helmicki, Jacobson and Nett. An algorithm using the Jackson polynomials is proposed that achieves an exponential convergence rate for exponentially stable systems. It is shown that this, and similar identification algorithms, can be successfully combined with a model reduction procedure to produce low-order models. Connections with the Nevanlinna-Pick interpolation problem are explored, and an algorithm is given in which the identified model interpolates the given noisy data. Some numerical results are provided for illustration. Finally, the case of unbounded random noise is discussed and it is shown that one can still obtain convergence with probability 1 under natural assumptions.	10A good example (but too similar to others for use in report)
65	J.R. Partington, 1992. "Robust Identification in H_∞." *Journal of Mathematical Analysis and Applications*, **166**, 1992, 428-441.				
66	J.R. Partington, 1993. "Algorithms for Identification in H_∞ with Unequally Spaced Function Measurements." *International Journal of Control*, **58**(1), July 1993, 21-31.			Worst-case identification in H_∞ is considered in the situation in which corrupted frequency response measurements are available at an arbitrary set of frequencies. Two new classes of algorithms are presented: one yields polynomial models directly, the other is a two-stage algorithm producing rational models. Each has improved convergence rates for the class of exponentially stable discrete-time systems (with errors typically $O(\varepsilon)+O(\Delta^r)$ for arbitrarily large r, where ε is the noise level and Δ is the maximum spacing between identification points). A numerical example is given.	7B modifications so that the frequency data may be arbitrarily spaced

#	Authors	Title	Source	Abstract	Notes
67	J.R. Partington and P.M. Mäkilä, 1995a. "Worst-Case Analysis of the Least-Squares Method and Related Identification Methods." *Systems and Control Letters*, **24**(3), Feb 13., 1995, 193-200.			We consider worst-case analysis of system identification under less restrictive assumptions on the noise than the l_∞ bounded error condition. It is shown that the least-squares method has a robust convergence property in l_2 identification, but lacks a corresponding property in l_1 identification (as well as in all other non-Hilbert space settings). The latter result is in stark contrast with typical results in asymptotic stochastic analysis of the least-squares method. Furthermore, it is shown that the Khintchine inequality is useful in the analysis of least l_p identification methods.	7B examine stochastic/ worst-case stuff
68	J.R. Partington and P.M. Mäkilä, 1995b. "Analysis of Linear Methods for Robust Identification in l_1." *Automatica*, **31**(5), May 1995, 755-758.			We consider worst-case analysis of system identification by means of the linear algorithms such as least-squares. We provide estimates for worst-case and average errors, showing that worst-case robust convergence cannot occur in the l_1 identification problem. The case of periodic inputs is also analysed. Finally a pseudorandomness assumption is introduced that allows more powerful convergence results in a deterministic framework.	7B l_1 identification; linear algorithm not robustly convergent; no nonlinear algorithm given
69	R. Pintelon and J. Schoukens.	Robust Identification of Transfer Functions in the *s*- and *z*-Domains	*IEEE Transactions on Instrumentation and Measurement*, **39**(4), Aug. 1990, 565-573.	A frequency-domain maximum likelihood estimator (MLE) to estimate the transfer function of linear continuous-time systems has already been developed in [1]. It assumes independent Gaussian noise on both the input and the output coefficients. In this paper, these results are extended to linear discrete-time systems. It is demonstrated that most of the properties of the estimator remain unchanged when it is applied to measured input and output Fourier coefficients corrupted with non-Gaussian errors. A robust Gaussian frequency-domain estimator results from this. It is very useful for the practical identification of linear systems. The theoretical results are verified by simulations and experiments.	0d maximum likelihood ID for discrete-time systems with non-Gaussian noise
70	B. Priel, E. Soroka, and U. Shaked, 1991. "The Design of Optimal Reduced-Order Stochastic Observers for Discrete-Time Linear Systems." *IEEE Transactions on Automatic Control*, **36**(11), Nov. 1991, 1300-1307. Reprinted in **36**(12), Dec. 1991, 1502-1509.			The minimum variance state estimation of linear discrete-time systems with random white noise input and partially noisy measurements is investigated. An observer of minimal order is found which attains the minimum-variance estimation error. The structure of this observer is shown to depend strongly on the geometry of the system. This geometry dictates the length of the delays that are applied on the measurements in order to obtain the optimal estimate. The transmission properties of the observer are investigated for systems that are left invertible, and free of measurement noise. An explicit expression is found for the transfer function matrix of this observer, from which a simple solution to the linear discrete-time singular optimal filtering problem is obtained.	0d
71	D. E. Rivera, J. F. Pollard, and C. E. García	Control-Relevant Prefiltering: a Systematic Design Approach and Case Study.	*IEEE Transactions on Automatic Control*, **37**(7), July 1992, 964-974.	System identification is the most demanding and time consuming step in the implementation of advanced control in the refining and petrochemical industries. As a result, *control-relevant* identification, which views the identification problem in terms of its impact on control system design, is a topic that possesses significant practical importance. In this paper, we specifically examine the use of control-relevant prefiltering applied to parameter estimation using prediction-error methods. The prefiltering step ensures that the estimated model retains those plant characteristics that are most significant with regards to the user's control requirements. We describe how to systematically build the prefilter in terms of the estimated model structure, the desired closed-loop speed-of-response, and the setpoint/disturbance characteristics of the control problem. Two implementation algorithms are presented which are applied to the plant data obtained from a distillation column. The results show that substantial improvements are obtained from control-relevant prefiltering in output error and partial least-squares estimation, while some caution must be exercised when applied to FIR and low-order ARX estimation.	0d control-relevant prefiltering
72	R. J. P. Schrama	Accurate Identification for Control: the Necessity of an Iterative Scheme.	*IEEE Transactions on Automatic Control*, **37**(7), July 1992, 991-994.	If approximate identification and model-based control design are used to accomplish a high-performance control system, then the two procedures have to be treated as a joint problem. Solving this joint problem by means of separate identification and control design procedures practically entails an iterative scheme. A frequency-response identification technique and a robust control design method are used to set up such an iterative scheme. Its utility is illustrated by an example.	0d ID/control integration
73	U. Shaked and Y. Theodor	A Frequency Domain Approach to the Problems of H_∞-Minimum Error State Estimation and Deconvolution.	*IEEE Transactions on Signal Processing*, **40**(12), Dec. 1992, 3001-3011.	The properties of the minimum H_∞-norm filtering estimation error are investigated, and the relation between the optimal estimator and the equalizing solution to the standard H_∞-minimization problem is discussed. The optimal estimation method is applied in the multivariable deconvolution problem. A simple deconvolution filter of minimum order is obtained which minimizes the H_∞-norm of the deconvolution error. The proposed methods of optimal estimation and deconvolution are useful in cases where the statistics of the disturbance and the noise signals are not completely known, or in cases where it is required to minimize the maximum singular value of the estimation, or the deconvolution, error spectrum.	0d state estimation in H_∞

#	Authors	Title	Source	Abstract	Notes
74	S. Shats and U. Shaked	Discrete-Time Filtering of Noise Correlated Continuous-Time Processes: Modeling and Derivation of the Sampling Period Sensitivities.	*IEEE Transactions on Automatic Control*, 36(1), Jan. 1991, 115-119.	The optimal discrete-time state estimation of continuous-time processes whose measurements are corrupted by additive white noise is considered in the case where the measurements are prefiltered by an integrator between sampling times. A discrete-time equivalent model, in which the measurements are written as a function of the state vector at the same instant, is developed for the general case where the continuous-time measurement and process noise signals are correlated. The equations governing the optimal filter, which is based on the discrete-time equivalent model, are presented. The properties of this filter are investigated model, are presented. The properties of this filter are investigated, in the case of a short sampling period, by deriving the first coefficients of the Maclaurin's expansions of the optimal gain and the error covariance matrices in powers of the sampling period. The obtained results are compared to the corresponding expressions that have been previously derived for the sampled-data regulator.	0d
75	D. S. Shook, C. Mohtadi, and S. L. Shah	A Control-Relevant Identification Strategy for GPC.	*IEEE Transactions on Automatic Control*, 37(7), July 1992, 975-980.	This note addresses the question of a suitable "control-relevant identification" strategy for a class of long-range predictive controllers. It is shown that under certain conditions the best process model for predictive control is that which is estimated using an identification objective function that is a dual of the control objective function. The resulting nonlinear least squares calculation is asymptotically equal to a standard recursive least squares with an appropriate (model and controller-dependent) FIR data prefilter. Experimental results demonstrate the validity and practicality of the proposed estimation law.	0d
76	R. S. Smith and J. C. Doyle	Model Validation: a Connection Between Robust Control and Identification.	*IEEE Transactions on Automatic Control*, 37(7), July 1992, 942-952.	Modern robust control synthesis techniques aim at providing robustness with respect to uncertainty in the form of both additive noise and plant perturbations. On the other hand, the most popular system identification methods assume that all uncertainty is in the form of additive noise. This has hampered the application of robust control methods to practical problems. This paper begins to address the gap between the models used in control synthesis and those obtained from identification experiments by considering the connection between uncertain models and data. The model validation problem addressed here is: given experimental data and a model with both additive noise and norm-bounded perturbations, is it possible that the model could produce the observed input-output data. This problem is studied for the standard H_∞/μ framework models. A necessary condition for such a model to describe an experimental datum is obtained. Furthermore, for a large class of models, in the robust control framework, this condition is computable as the solution of a quadratic optimization problem.	5b good background
77	T. Söderström and K.J. Åström, 1995. "Special Issue on Trends in System Identification." *Automatica*, 31(12), Dec. 1995, 1689-1690.			*(no abstract)*	5c
78	B. Wahlberg and L. Ljung	Hard Frequency-Domain Model Error Bounds From Least-Squares Like Identification Techniques.	*IEEE Transactions on Automatic Control*, 37(7), July 1992, 900-912.	The problem of deriving so-called hard error bounds for estimated transfer functions is addressed. A hard bound is one that is sure to be satisfied, i.e., the true systems Nyquist plot will be confined with certainty to a given region, provided that the underlying assumptions are satisfied. By blending *a priori* knowledge and information obtained from measured data, we show how the uncertainty of transfer function estimates can be quantified. The emphasis is on errors due to model mismatch. The effects of unmodeled dynamics can be considered as bounded disturbances. Hence, techniques from set membership identification can be applied to this problem. The approach taken corresponds to weighted least-squares estimation, and provides hard frequency-domain transfer function error bounds. The main assumptions that are used in the current contribution are: that the measurement errors are bounded; that the true system is indeed linear with a certain degree of stability; and that there is some knowledge about the shape of the true frequency response.	0d
79	W. Wang and M. G. Safonov	Relative-Error H_∞ Identification From Autocorrelation Data — A Stochastic Realization Method.	*IEEE Transactions on Automatic Control*, 37(7), July 1992, 1000-1004.	A variant on the balanced stochastic truncation (BST) method for approximate realization of power spectrum matrices is shown to form the basis for an identification procedure that is well-suited to the task of determining relative-error-bounded approximate plant models for use in control design from input-output cross correlation data. Central to the theory is a new l_∞-norm bound on the relative-error between an exact realization of the data and a BST approximate realization.	0d
80	D. Xiong, G. Gu, and K. Zhou	Identification in H_∞ via Convex Programming	*1993 American Control Conference*, San Francisco, California, June 2-4, 1993. Proceedings (American Automatic Control Council, Evanston, Illinois), 1537-1538.	Convex programming techniques is employed to solve the problem of system identification in H_∞ which was first formulated in (Helmicki, Jacobson, and Nett, *IEEE Trans. Auto. Contr.*, 36, 1991). A unique feature of this proposed identification algorithm is that it has a performance close to that of a Nevanlinna Pick interpolation based algorithm as studied in [1,5] which is strongly optimal within a factor of two. An explicit bound is also derived for the worst case identification error measured in the H_∞ norm.	5B

#	Authors	Title	Source	Abstract	Notes
81	I. Yaesh and U. Shaked.	Game Theory Approach to Optimal Linear Estimation in the Minimum H_∞-Norm Sense.	*28th IEEE Conference on Decision and Control*, Tampa, Florida, Dec. 13-15, 1989. Proceedings (IEEE, New York), 421-425.	A game theory approach is presented to optimal state estimation. It is found that under certain conditions a min-max estimation is identical to the optimal estimation in the minimum H_∞-norm sense. These conditions are similar to those obtained in (Mintz, "A Kalman filter as minimax estimator", J. Opt. Theory and Appl., **9**, 1972, 99-111), where the relation between Kalman filtering and the min-max terminal state estimation has been explored. This new interpretation of the H_∞-optimal state estimation provides a better insight into the mechanism of H_∞-optimal filtering.	4b
82	I. Yaesh and U. Shaked	Two-Degree-of-Freedom H_∞-Optimization of Multivariable Feedback Systems.	*IEEE Transactions on Automatic Control*, **36**(11), Nov. 1991, 1272-1276.	A solution is derived to the two-degree-of-freedom H_∞-minimization problem that arises in the design of multivariable optimal continuous-time stochastic control systems. A decoupling approach is applied, which enables a partially independent design of the prefilter and the feedback controller and yields a simple solution to the optimization problem. This solution is obtained by transforming the optimization problem into two standard form ("four-block") problems.	0d
83	I. Yaesh and U. Shaked	Nondefinite Least Squares and its Relation to H_∞-Minimum Error State Estimation.	*IEEE Transactions on Automatic Control*, **36**(12), Dec. 1991, 1469-1472.	The problem of recursive nondefinite least squares state estimation of continuous-time stationary processes is solved, by applying variational calculus. A comparison of the derived solution to the result that is obtained for the H_∞-minimum error estimation suggests a new interpretation for the H_∞-optimal estimation mechanism. According to this interpretation, the estimator tries to optimally estimate the required combination of the states, in the l_2-norm sense, against the worst disturbance signal that stems from a fictitious measurement of this combination.	5b
84	I. Yaesh and U. Shaked, 1992. "Game Theory Approach to Optimal Linear State Estimation and its Relation to the Minimum H_∞-Norm Estimation." *IEEE Transactions on Automatic Control*, **37**(6), June 1992, 828-831.			A possible game theory approach to optimal state estimation is presented. It is found that in a certain differential game, the minimizer's policy is identical to the one obtained by optimal estimation in the minimum H_∞-norm sense. This interpretation of the H_∞-optimal state estimation provides a better insight into the mechanism of the H_∞-optimal filtering, especially, in the case where the exogenous signals are not energy bounded.	4b
85	I. Yaesh and U. Shaked	Game Theory Approach to State Estimation of Linear Discrete-Time Processes and its Relation to H_∞-Optimal Estimation.	*International Journal of Control*, **55**(6), June 1992, 1443-1452.	A game theory approach to the state-estimation of linear discrete-time systems is presented. The resulting state estimation suggests an alternative to the Kalman filter, in cases where the exact statistics of the input and the measurement noise processes is not known. It turns out that the game-theoretic filter provides an H_∞-optimal estimation. Moreover, it is shown that the covariance matrix of the estimation error is bounded, from above, by the solution of a modified Riccati equation.	4b
86	N. Young, 1988. *An Introduction to Hilbert Space*, Cambridge University Press (Cambridge, England), 1988.			*(no abstract)*	10a solution to the Nehari problem
87	G. Zames, 1979. "On the Metric Complexity of Causal Linear Systems: ε -entropy and ε -dimension for Continuous-Time." *IEEE Transactions on Automatic Control*, **24**(2), April 1979, 222-230.			Estimates of ε -entropy and ε -dimension in the Kolmogorov sense are obtained for a class of causal, linear, time-invariant, continuous-time systems under the assumptions that impulse responses satisfy an exponential order condition $\lvert f(t)\rvert \le Ce^{-at}$, and frequency responses satisfy an attenuation condition $\lvert F(j\omega)\rvert \le K\omega^{-1}$. The dependence of ε -entropy and ε -dimension on the accuracy ε is characterized by order, type, and power indexes. Similar results for the discrete-time case are reviewed and compared.	5b one of the earliest references to the bounded deterministic approach; uses the theory of metric complexity to study issues related to the complexity of identification

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1997	Contractor Report

4. TITLE AND SUBTITLE

On-Line Modal State Monitoring of Slowly Time-Varying Structures

5. FUNDING NUMBERS

529-50-04-00-RR-00-000

6. AUTHOR(S)

Erik A. Johnson, Lawrence A. Bergman, and Petros G. Voulgaris

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Illinois at Urbana-Champaign
Department of Aeronautical and Astronautical Engineering
104 South Wright Street
Urbana, Illinois 61801

8. PERFORMING ORGANIZATION REPORT NUMBER

H-2202

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

NASA Dryden Flight Research Center
Edwards, California

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

CR-198057

11. SUPPLEMENTARY NOTES

Technical Monitor: Lawrence C. Freudinger, NASA Dryden. NASA Contract NAG 2-4001.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified—Unlimited
Subject Category 05

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Monitoring the dynamic response of structures is often performed for a variety of reasons. These reasons include condition-based maintenance, health monitoring, performance improvements, and control. In many cases the data analysis that is performed is part of a repetitive decision-making process, and in these cases the development of effective on-line monitoring schemes help to speed the decision-making process and reduce the risk of erroneous decisions. This report investigates the use of spatial modal filters for tracking the dynamics of slowly time-varying linear structures. The report includes an overview of modal filter theory followed by an overview of several structural system identification methods. Included in this discussion and comparison are H-infinity, eigensystem realization, and several time-domain least squares approaches. Finally, a two-stage adaptive on-line monitoring scheme is developed and evaluated.

14. SUBJECT TERMS

Modal filtering, Reciprocal modal vector, Parameter estimation, Structural dynamics, On-line health monitoring

15. NUMBER OF PAGES

188

16. PRICE CODE

A09

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited